# PointerGuess: Targeted Password Guessing Model Using Pointer Mechanism

Kedong Xiu and Ding Wang, *Nankai University*

https://www.usenix.org/conference/usenixsecurity24/presentation/xiu

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

# PointerGuess: Targeted Password Guessing Model Using Pointer Mechanism

Kedong Xiu, Ding Wang
College of Cyber Science, Nankai University, Tianjin, China; {xiukedong, wangding}@nankai.edu.cn
Key Laboratory of Data and Intelligent System Security (NKU), Ministry of Education, Tianjin, China
Tianjin Key Laboratory of Network and Data Security Technology, Nankai University, Tianjin, China

## Abstract

Most existing targeted password guessing models view users' reuse behaviors as sequences of edit operations (e.g., insert and delete) performed on old passwords. These atomic edit operations are limited to modifying one character at a time and cannot fully cover users' complex password modification behaviors (e.g., modifying the password structure). This partially leads to a significant gap between the proportion of users' reused passwords and the success rates that existing targeted password models can achieve. To fill this gap, this paper models users' reuse behaviors by focusing on two key components: (1) What they want to copy/keep; (2) What they want to tweak. More specifically, we introduce the pointer mechanism and propose a new targeted guessing model, namely PointerGuess. By hierarchically redefining password reuse from both personal and population-wide perspectives, we can accurately and comprehensively characterize users' password reuse behaviors. Moreover, we propose MS-PointerGuess, which can employ the victim's multiple leaked passwords.

By employing 13 large-scale real-world password datasets, we demonstrate that PointerGuess is effective: (1) When the victim's password at site $A$ (namely $pw_A$) is known, within 100 guesses, the average success rate of PointerGuess in guessing her password at site $B$ (namely $pw_B$, $pw_A \neq pw_B$) is 25.21% (for common users) and 12.34% (for security-savvy users), respectively, which is 21.23%∼71.54% (38.37% on average) higher than its foremost counterparts; (2) When not excluding identical password pairs (i.e., $pw_A$ can equal $pw_B$), within 100 guesses, the average success rate of PointerGuess is 48.30% (for common users) and 28.42% (for security-savvy users), respectively, which is 6.31%∼15.92% higher than its foremost counterparts; (3) Within 100 guesses, the MS-PointerGuess further improves the cracking success rate by 31.21% compared to PointerGuess.

## 1 Introduction

Textual passwords stubbornly survive as the most prevalent authentication method, and they are unlikely to be replaced in the foreseeable future, because none of their alternatives (e.g., biometric authentication [8, 22], multi-factor authentication [46, 47], and graphical passwords [6]) can compete with textual passwords in terms of simplicity to use, easiness to change and low cost to deploy [9, 10, 30, 37].

Recent research [11, 31, 41] shows that the average user has 80∼107 distinct online accounts, and a large fraction of investigated users (58%∼79% [13, 18, 38, 53]) tend to reuse their passwords across sites, even though 91% of them are aware of the risks associated with password reuse [25]. To address this issue, security experts [12, 43] recommend using password managers to help users create secure passwords and protect users from password guessing attacks. A password manager is designed to store a user's passwords, generate secure passwords, and identify any weak or compromised passwords [29]. However, due to users' lack of trust in password managers and the fact that it is too risky to store all passwords in one place [36, 42, 63], most users still tend to manage their password by themselves. This indicates that users' password reuse behaviors remain a significant security vulnerablity.

On the other hand, unending large-scale password dataset breaches (e.g., [1, 4, 16, 17, 34]) provide attackers with ample training data to conduct targeted guessing attacks. Still, recent studies show that two-thirds of users "modify passwords in a non-trivial way" [56, 57]. This suggests that designing an effective targeted guessing model to accurately characterize users' reuse behaviors is not an easy task.

### 1.1 Motivations and design challenges

Recent research shows that 58%∼79% of investigated users directly reuse or simply modify their existing passwords [13, 18, 38, 53]. However, the cracking success rates that state-of-the-art targeted password guessing models (i.e., Pass2Edit [57] and Pass2Path [38]) can achieve are much lower than this statistic (e.g., within 1,000 guesses, Pass2Edit [57] achieves a cracking success rate of 52.01% for common users, which is 11.52%∼51.89% lower than the reported statistic). This indicates that existing targeted password models may not effectively characterize users' reuse behaviors, and the threat of password reuse guessing attacks might be underestimated.

Despite numerous string similarity metrics (e.g., edit distance and cosine similarity), they fail to measure users' password reuse behaviors comprehensively. For instance, while modifying `IloveMacOP` to `MacOP6789`, neither edit distance (employed by Pass2Path [38]) nor cosine similarity (employed by Pass2Edit [57]) can accurately measure their similarity due to overlooking complex reuse behaviors. This highlights a gap in existing research on password reuse, indicating a lack of an appropriate definition for "password reuse". In all, how to accurately characterize users' reuse behaviors given limited guessing attempts (e.g., 100 as recommended by NIST [19]) remains a challenging problem. Here we explain why.

First, characterizing the transformation rules that users employ to modify their passwords is quite subtle. In general, we can view the process of users reusing their old password as involving two major components: (1) Identifying what the user wants to copy from her old password; and (2) Determining what the user wants to tweak or generate based on her old password. However, most existing targeted password models (e.g., Pass2Path [38] and Pass2Edit [57]) focus primarily on the second component. These state-of-the-art models characterize the new password modification process as a sequence of atomic edit operations (i.e., deleting, inserting, or substituting *one character at a time*). Then, they predict the sequence of edit operations in a "password-to-path" task. For example, suppose we modify a user's old password $pw_A$=`IloveMacOP` to $pw_B$=`MacOP6789`, then the edit operation sequence from $pw_A$ to $pw_B$ is {`<BOS>`, (`Del`, 0, 'I'), (`Del`, 1, 'l'), (`Del`, 2, 'o'), (`Del`, 3, 'v'), (`Del`, 4, 'e'), (`Ins`, 10, '6'), (`Ins`, 10, '7'), (`Ins`, 10, '8'), (`Ins`, 10, '9'), `<EOS>`}, where (`Del`, 0, 'I') denotes deleting the character 'I' in the first position of $pw_A$, and `<BOS>`/`<EOS>` represents the start/end of the edit sequence.

To be effective, Pass2Edit [57] and Pass2Path [38] have to define a large number of atomic edit operations (e.g., Pass2Edit [57] defines a total of 1,561 atomic operations). Besides, they filter out "dissimilar" password pairs when training (e.g., Pass2Path [38] only utilizes password pairs with edit distance≤4 for training to avoid the negative impacts of futile/distant password pairs like `yjqqq916198` → `916198yj`). This makes it difficult for them to generate long (yet realistic) edit sequences (e.g., edit distance≥5), overlooking users' macroscopic population-wide reuse behaviors (e.g., using popular passwords and substituting long segments). To mitigate this defect, they resort to heuristic approaches to combine a popular password dictionary with the generated guesses.

Second, as 49%∼65% of websites [5, 32] adopt security mechanisms (e.g., account lockout and login throttling as recommended by NIST [60]) to resist online guessing, the guess number allowed is often very small. For instance, the Alexa top-10 websites allow 120∼1,140 attempts per day, i.e., 3,600∼43,200 attempts per month [54]. As the possible password space is large, it is challenging to prioritize password reuse behaviors in a personalized manner under such small

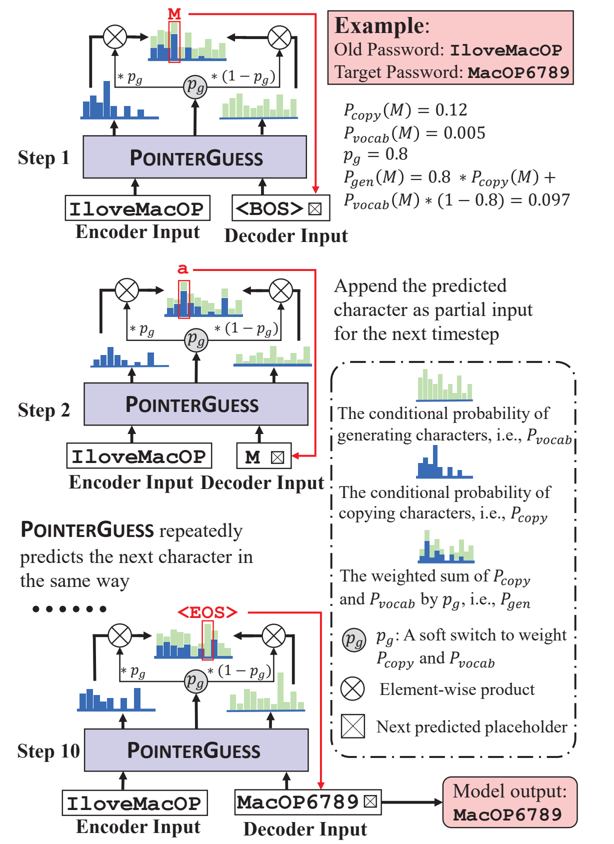



Figure 1: An example of POINTERGUESS generating guesses. Suppose the old password is `IloveMacOP`, the target password is `MacOP6789`. At each timestep, POINTERGUESS generates two conditional probabilities, $P_{copy}$ and $P_{vocab}$, respectively. `<BOS>`/`<EOS>` represents the start/end of the generation.

guess numbers. In all, building a password reuse-based guessing model that can accurately capture users' comprehensive password reuse behaviors remains a challenging task.

To address these issues, we propose a new targeted guessing model, POINTERGUESS (see Fig. 1 for a high-level view), directly taking password characters as input and output, to avoid the need for defining numerous atomic edit operations. Additionally, unlike [38, 57], it trains on the entire dataset without filtering out any distant password pairs. By using the pointer mechanism [50], POINTERGUESS figures out how likely it is to copy characters from the old password and how likely it is to create new characters, enabling POINTERGUESS to capture both personal and population-wide reuse behaviors (e.g., using popular passwords and substituting long segments). Particularly, within 100 guesses, POINTERGUESS achieves a success rate that is 38.18% on average higher than its counterparts. Furthermore, we propose MS-POINTERGUESS for attack scenarios with the victim's multiple leaked passwords. Experimental results demonstrate that MS-POINTERGUESS outperforms POINTERGUESS in practical scenarios.

## 1.2 Related work

The first targeted guessing algorithm based on password reuse was proposed by Das et al. at NDSS'14 [14]. They introduced a heuristic algorithm that applies eight transformation rules (e.g., insertion and deletion) to the old password of a victim in a predetermined order to generate guesses. While this algorithm demonstrates superior performance compared to some trawling guessing algorithms like PCFG [59], it remains entirely heuristic in nature. Its fundamental limitation is that it uses the same transformation rules across all users, lacking consideration for personalized rule priority.

At CCS'16, Wang et al. [56] proposed TarGuess-II based on the PCFG algorithm. As the first probabilistic-based targeted password model, its key idea is that the user performs only one operation (e.g., insertion, deletion) on her old password or password structure once at a time. It analyzes the transformation path between the password pairs to learn the probability of the corresponding transformation. It could output guesses in descending order of probability when generating guesses.

At IEEE S&P'19, Pal et al. [38] proposed a password reuse model based on deep learning, named Pass2Path. It utilizes a seq2seq model [49] and conceptualizes its task as predicting the edit-operation path from the old password to the new password. Pass2Path can intuitively predict the edit operations and accurately generate guesses.

One limitation of Pass2Path [38] is that it can not capture the impact between the editing operations and the corresponding editing effects. Accordingly, Wang et al. [57] proposed a new algorithm called Pass2Edit. Unlike Pass2Path, Pass2Edit models the new password generation task as a classification task and uses a multi-step decision-making training mechanism to capture users' reuse behaviors. However, both Pass2Edit and Pass2Path use *atomic* edit operations, once at a time and fiter out "dissimilar" password pairs during training, and thus they can not model transformation operations on long segments effectively.

At USENIX Security'23, Wang et al. [58] introduced RFGuess-reuse, a targeted password guessing model. It represents password prefixes as high-dimensional vectors and employes a random forest classifier to predict each edit operation for each type of string. Results show that RFGuess-reuse performs comparably to TarGuess-II [56] and Pass2Path [38].

### 1.3 Our contributions

We summarize our main contributions as follows:

- **A new targeted guessing model**. We introduce the pointer mechanism into the password reuse research domain and propose a new targeted password guessing model, PointerGuess. By leveraging the pointer mechanism, PointerGuess can effectively identify what the user wants to copy/keep and what the user wants to tweak from the old password. Furthermore, considering the increasingly realistic scenario of multiple password leakage for common users, we propose a brand-new targeted guessing model, MS-PointerGuess, to assess the threat of attackers using multiple old passwords to compromise the target password. By hierarchically redefining "password reuse" on two levels, we demonstrate the effectiveness of PointerGuess and provide a new angle to understand the performance of existing models.
- **Extensive evaluation**. We demonstrate the effectiveness of PointerGuess on 12 practical attack scenarios by employing 11 large-scale password datasets. More specifically, within 100 guesses, PointerGuess outperforms the state-of-the-art models by 38.17% on average, without counting identical password pairs and mixing an extra popular password dictionary. Furthermore, we demonstrate the superior performance of MS-PointerGuess over our PointerGuess in two practical attack scenarios. More specifically, within 100 guesses, the MS-PointerGuess achieves a success rate 31.21% (on average) higher than PointerGuess.
- **A password reuse-based password strength meter**. We introduce a password reuse-based password strength meter, called PR-PSM, by integrating Zxcvbn [60] with PointerGuess to enhance the evaluation accuracy. Our experiments demonstrate the importance of considering password reuse attacks for improving personalized PSMs, and highlight the importance of avoiding password reuse for security-critical accounts.
- **Some insights**. Our analysis shows that PointerGuess can capture complex password reuse behaviors (e.g., `1991322322` → `1.99132E+12`). These findings enhance our understanding of password reuse and showcase the effectiveness of PointerGuess. Moreover, our results indicate that in multiple old password reuse attack scenarios, the target password is more likely to be found within old passwords, highlighting the increased risk of multiple password compromises against users.

## 2 Background

### 2.1 Modeling password guessing probability

There are two approaches for neural network-based models to compute the conditional password probability: (1) directly predicting the targeted password character sequence (e.g., PassTrans [21]); and (2) predicting the atomic edit operation sequence from the old password to the target password (e.g., Pass2Path [38] and Pass2Edit [57]). The first approach focuses on predicting the exact character sequence by modeling conditional probabilities of generating each character, and we can express the conditional probability $P(pw_B|pw_A)$ as

$$\begin{aligned} P(pw_B|pw_A) &= P\left(c_1^{'}, c_2^{'}, ..., c_M^{'}|c_1, c_1, ..., c_N\right) \\ &= \prod_{i=1}^{M} P\left(c_i^{'}|pw_A, c_{<i}^{'}\right), \end{aligned} \tag{1}$$

where $c_{<i}^{'} = (c_1^{'}, .., c_{i-1}^{'})$ denotes the subsequence of $pw_B$, and how to model $P\left(c_i^{'}|pw_A, c_{<i}^{'}\right)$ depends on the specific model.

The second approach aims to predict the atomic edit operation sequence needed to transform the old password into the targeted password. We denote $\tau_{A,B} = (e_1, e_2, ..., e_E)$ as the transformation path from $pw_A$ to $pw_B$. The conditional probability is modeled to estimate the likelihood of each edit operation given the old password. In this case, we can express the conditional probability $P(pw_B|pw_A)$ as

$$P(pw_B|pw_A) = P(\tau_{A,B}|pw_A) = \prod_{i=1}^{E} P(e_i|pw_A, e_{<i}), \tag{2}$$

where the $e_{<i}$ denotes the subsequence of $\tau_{A,B}$, which is $(e_1, .., e_{i-1})$, and the specific formula of the conditional probability $P(e_i|pw_A, e_{<i})$ depends on the specific model. Most recent targeted password guessing models (e.g., Pass2Path [38] and Pass2Edit [57]) are based on the second approach.

Predicting the atomic edit operation sequence can intuitively capture the transformations in password reuse, providing insights into the specific operations required. However, employing this approach requires defining a substantial number of atomic operations, which limits the model's ability to generate long/complex operation sequences (i.e., giving very low probabilities to such sequences). Thus, POINTERGUESS employs the first approach to model the conditional password probability in a novel manner. See details in Sec. 3.2.

## 2.2 Password similarity metrics

Generally, two main types of similarity metrics are commonly used to measure password similarity: syntactic metrics [20, 38, 57] and semantic metrics [14, 52]. Syntactic metrics calculate the "structural distance" (e.g., edit distance and cosine similarity) as a similarity score, while semantic metrics focus on capturing the structural and semantic similarity between passwords. For instance, Wang et al. [52] presented a workflow with eight rules (e.g., leet and reversal) to quantify users' password reuse behaviors.

In this paper, we primarily consider four representative syntactic metrics due to their simplicity and sufficiency for our use in latter sections:

**Spatial distance-based metrics.** These metrics measure the spatial distance between passwords, considering their structure/character-level differences, such as cosine similarity [57] and edit distance [27]. They primarily focus on the positional and directional differences or equally the number of operations needed to transform one password into another.

**Sequence alignment-based metrics.** These metrics align the character sequences of two passwords to identify common segments and measure the similarity based on the alignment, such as the Needleman-Wunsch algorithm [35] and the Largest Common Substring algorithm [14], which consider the order and position of characters in the sequences.

**Overlap-based metrics.** These metrics quantify the overlap or common strings between passwords, providing a similarity score based on the common characters (e.g., the Dice coefficient [15]). They focus on the common elements between

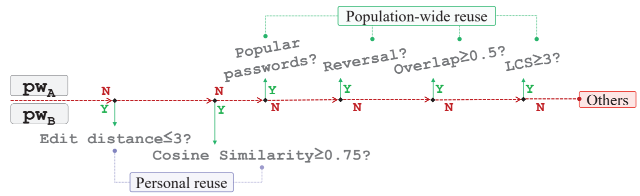



Figure 2: The workflow of detecting password reuse behaviors based on the new password reuse definition. "Others" refers to unmatched password pairs.

passwords rather than their structural differences.

**Combination metrics.** Combination metrics provide a comprehensive measurement of similarities by integrating multiple individual metrics. In the work by Guo et al. [20], a combination of edit distance and cosine similarity is employed to capture both structural and semantic aspects of password similarities. Edit distance focuses on the absolute positional dimension, quantifying the atomic operations (e.g., insertion, deletion, and substitution) needed to transform one password into another, while cosine similarity gauges syntactic resemblance by further considering the angle between vectors representing passwords in a high-dimensional space.

# 3 POINTERGUESS: A targeted password reuse guessing model based on pointer mechanism

In this section, first, we introduce a hierarchical definition of "password reuse", which provides a new angle to understand users' password reuse behaviors. Second, we describe our model, POINTERGUESS, and how to model the conditional password probability using POINTERGUESS. Third, we detail our model methodology and hyperparameters.

## 3.1 A new definition of password reuse

As mentioned in Sec. 1.1, existing targeted password guessing models (e.g., Pass2Path [38] and Pass2Edit [57]) have inherent limitations. They prefer to use syntactic metrics (e.g., edit distance [38] and cosine similarity [57]) to measure password similarities and evaluate their effectiveness in characterizing users' password reuse behaviors. To accurately capture users' password reuse behaviors, we propose a new definition of "password reuse" (combining both syntactic and semantic metrics, as depicted in Fig. 2), which hierarchically categorizes users' password reuse behaviors into two distinct levels: personal reuse and population-wide reuse.

**Personal reuse** refers to the simple modifications that users tend to apply to their old passwords based on their preferences and the characteristics of old passwords. These modifications typically involve a limited number of edit operations, such as adding/deleting the first/last character and/or replacing a character with visually similar alternatives (e.g., replacing `a` with `@`). These new passwords, primarily created through personal reuse, can be easily identified as instances of password reuse using syntactic metrics (e.g., edit distance [38]).

**Population-wide reuse** refers to some more complex and challenging-to-identify password reuse behaviors. In general, it relates to users' reuse patterns that can be observed across

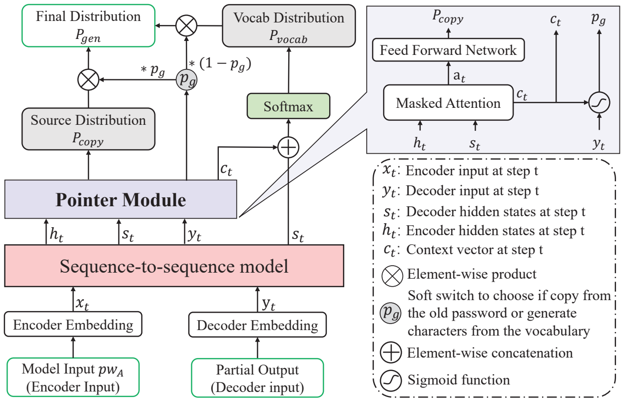



Figure 3: Model architecture of PointerGuess, which is based on [45, 50] and consists of a basic seq2seq model [49] and a pointer module. $P_{gen}$ (i.e., "Final Distribution") denotes the conditional password probability of PointerGuess generating the next character, $P_{copy}$ and $P_{vocab}$ are the conditional probabilities of copying characters from the old password, and of generating new characters from the vocabulary, respectively. PointerGuess employs a soft switch $p_g$ to decide whether to copy characters from the old password or generate new characters from the vocabulary.

the entire dataset, encompassing the reuse of popular passwords (which are frequently chosen by a substantial number of users, e.g., `KeveinMobile` $\rightarrow$ `password123`). Furthermore, it extends to reusing some specific popular password segments, like `KevinMobile` $\rightarrow$ `Kevin@gmail.com`.

## 3.2 Modeling conditional guessing probability

As discussed in Sec. 2.1, we adopt the directly predicting characters approach (i.e., Eq. 1) to address limitations in Pass2Edit [57] and Pass2Path [38]. We start with a basic sequence-to-sequence (seq2seq) model, utilizing it to directly model the similarity between users' target and old passwords. However, the basic seq2seq model may neglect the impact of low-frequency yet crucial characters in a given old password (i.e., the encoder input) on predicting subsequent characters.

For example, as shown in Fig. 1, the basic seq2seq model struggles to generate '`M`' as the first character due to the infrequent occurrence of '`M`' as the first character in the training set. More specifically, at the first step, it assigns an extremely low probability to '`M`' (i.e., $P_{vocab}(\texttt{M}) = 0.005$). This highlights the challenge of modeling the conditional password probability: *How to assign a sufficient likelihood to those crucial characters that appear in the given old password but have low frequency in the training set*.

To address this issue, we introduce the *Pointer Module* to add additional likelihood for those characters appearing in the old password. Our PointerGuess incorporates $P_{vocab}$ (the conditional probability of generating characters from vocabulary, i.e., 95 printable characters) and $P_{copy}$ (the conditional probability of copying from the old password), to model the conditional password probability ($P_{gen}$). As shown in Fig. 1, after incorporating $P_{copy}$, the probability of generating '`M`' significantly increases (i.e., $P_{gen}(\texttt{M}) = 0.097 \gg P_{vocab}(\texttt{M}) = 0.005$). In each subsequent step, PointerGuess dynamically adjusts the probabilities of each character through $P_{copy}$ and $P_{vocab}$, gradually approaching the correct target password (i.e., `MacOP6789`). Below, we formally describe how PointerGuess models the conditional password probability.

As shown in Fig. 3, at each timestep $t$, PointerGuess uses $pw_A$ as the input to produce encoder input $x_t$, and uses the previously generated character sequence as decoder input $y_t$. The basic seq2seq model of PointerGuess outputs encoder hidden states $h_t$ and decoder hidden states $s_t$. Subsequently, PointerGuess computes the conditional probability of generating characters from the vocabulary, i.e., $P_{vocab}$, as

$$P_{vocab} = softmax\left(W'\left(W * [s_t, c_t] + b_{out}\right) + b'_{out}\right), \quad (3)$$

where $[s_t, c_t]$ denotes concatenating $s_t$ and $c_t$, $W$, $W'$, $b_{out}$, $b'_{out}$ are learnable parameters, and $c_t$ is the context vector at timestep $t$. $c_t$ represents the context information learned from the encoder input (i.e., $pw_A$) at timestep $t$.

Additionally, PointerGuess employs the *Pointer Module* to capture user-specific patterns and reuse behaviors, producing the conditional probability of copying characters from the old password, i.e., $P_{copy}$, which can be expressed as

$$P_{copy}(c) = FFN\left(\sum_{j:c_j=c} a_j^t\right), \quad (4)$$

where $FFN(\cdot)$ is a feed-forward network used to rescale the attention vector $a^t$ generated by the pointer module at timestep $t$. If the input sequence does not contain the token $c$, then the value of $P_{copy}(c)$ is zero.

To facilitate flexible decision-making on *whether to copy characters from the old password or generate new characters from the vocabulary*, PointerGuess utilizes a soft switch $p_g$, which is

$$p_g = \sigma\left(W_c * c_t + W_s * s_t + W_y * y_t + b_g\right), \quad (5)$$

where $W_c$, $W_s$, $W_y$, $b_g$ are learnable parameters. $y_t$ is the decoder input at timestep $t$ and $\sigma(\cdot)$ is a sigmoid function.

Finally, PointerGuess integrates $P_{vocab}$ and $P_{copy}$ to generate $P_{gen}$, representing the conditional password probability $P_{gen}$, which is expressed as

$$P_{gen}(c) = p_g * P_{copy}(c) + (1 - p_g) * P_{vocab}(c). \quad (6)$$

This allows PointerGuess to dynamically decide between copying characters from the old password and generating new characters from the vocabulary.

## 3.3 Methodology and configuration

As shown in Fig. 4, PointerGuess consists of three phases: Preprocess, training, and generation. We now present the workflow of PointerGuess that tackles model training and guess generation, and detail the preprocess phase in Sec. 4.

**Model architecture**. As shown in Fig. 3, our PointerGuess mainly consists of a sequence-to-sequence model (with an encoder and a decoder) and an extra *Pointer Module* [50]. The encoder is a 1-layer Bi-LSTM, which is used to capture the

Table 1: Data cleaning of 13 password datasets leaked from various web services ("PWs" stands for passwords).

| Dataset | Language | Leaked Time | Original PWs | Unique PWs | Removed% | Email invalid | PW invalid | After cleaning | Service |
|---|---|---|---|---|---|---|---|---|---|
| 000Webhost | English | Oct. 2015 | 15,299,907 | 10,526,769 | 0.76% | 195 | 67,401 | 15,183,627 | Web hosting |
| LinkedIn | English | Jan. 2012 | 54,656,615 | 34,282,741 | 0.23% | 0 | 122,051 | 54,534,564 | Job hunting |
| Yahoo | English | Jul. 2012 | 5,737,798 | 3,495,654 | 0.95% | 118 | 54,105 | 5,683,574 | Portal |
| RedMart‡ | English | Oct. 2020 | 1,108,774 | — | 0.0% | 0 | — | 1,108,774 | E-commerce |
| ClixSense | English | Jun. 2016 | 2,222,045 | 1,627,069 | 0.07% | 0 | 1,445 | 2,220,600 | E-commerce |
| LiveAuctioneers | English | Jun. 2020 | 2,912,377 | 2,229,358 | 1.34% | 3,341 | 35,646 | 2,876,496 | Online auction platform |
| Tianya | Chinese | Dec. 2011 | 30,816,592 | 12,873,222 | 0.03% | 5,783 | 3,279 | 30,807,530 | Social forum |
| 126 | Chinese | Dec. 2011 | 6,392,568 | 3,764,740 | 0.24% | 0 | 14,995 | 6,377,573 | Email |
| Dodonew | Chinese | Dec. 2011 | 16,282,286 | 10,010,744 | 1.57% | 42,585 | 30,085 | 16,026,270 | E-commerce |
| Taobao | Chinese | Feb. 2016 | 15,072,418 | 11,633,759 | 0.01% | 1,176 | 90 | 15,071,153 | E-commerce |
| CSDN | Chinese | Dec. 2011 | 6,428,410 | 4,034,779 | 0.05% | 5 | 3,157 | 6,425,246 | Programmer forum |
| 4iQ | Mixed | Dec. 2017 | 1,400,553,869 | 445,259,097 | 1.36% | 575,283 | 18,475,938 | 1,381,502,648 | Unkonwn |
| COMB | Mixed | Feb. 2021 | 3,279,064,312 | 855,833,811 | 2.97% | 81,542,117 | 15,718,941 | 3,181,803,254 | Unkonwn |

‡RedMart passwords are in salted-hash, and we use them as real targets in attack scenario #8 (000Webhost $\rightarrow$ RedMart). See more details in Table 2.

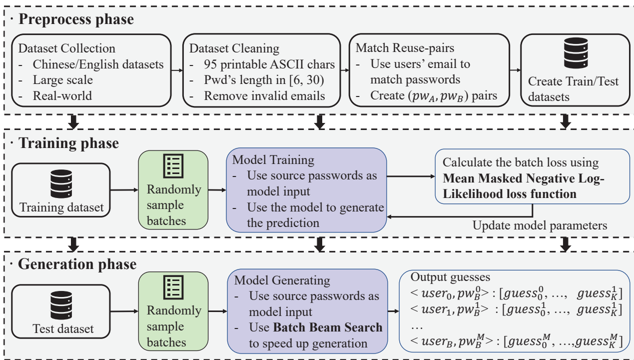



Figure 4: The workflow of POINTERGUESS. There are three phases: Preprocess, Training, and Generation. Specifically, $M$ and $K$ denote the number of users per batch and Top-$K$ password guesses for each user, respectively.

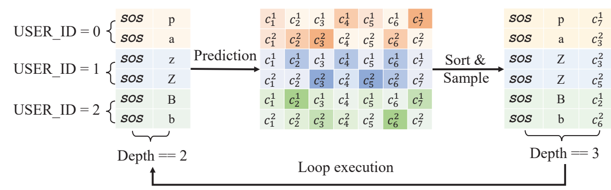



Figure 5: An example of batch-beam search.The batch size is three and beam size is two. ***SOS*** is the start symbol. Darker color means higher probability. For example, the second user (i.e., `USER_ID = 1`) choose $c_3^2$ and $c_5^2$ (which denote the predicted characters) due to their high probabilities. Guess generation will continue until all users are completed.

contextual information of the old password. The decoder is a 1-layer Bi-LSTM used to generate conditional guesses based on the captured contextual information from the encoder. We set the hidden dimension of encoder and decoder as 128. We add an extra *reduce layer* to process the output of the encoder and decoder, which is used to aggregate the encoder's output and further improve the performance of our model.

Additionally, we integrate a *Pointer Module* [50] into our POINTERGUESS, comprising an attention network and a feed-forward network. The attention network highlights relevant parts of the old password during model decoding. It generates the attention vector and employs a sigmoid function to yield the context information vector at each timestep.

**Training phase.** During this phase, POINTERGUESS randomly samples batches of password pairs $(X_{BS}, Y_{BS})$ from the training set, where $X_{BS}$ represents the old passwords, and $Y_{BS}$ represents the corresponding target passwords. Our model inputs the old passwords and generates guesses at the character level, denoted as $\hat{Y}_{BS}$. The training objective involves a loss function, denoted as $\mathcal{L}$, which measures the log probability of $\hat{Y}_{BS}$ being aligned with the ground truth passwords $Y_{BS}$. The goal of $\mathcal{L}$ is to find the optimized parameters $\theta^*$, which are

$$\theta^* = \arg\min_{\theta} \left( \mathcal{L}_\theta \left( Y_{BS}, \hat{Y}_{BS} \right) \right), \tag{7}$$

where $\theta$ denotes model parameters. We use Mean Masked Negative Log-Likelihood as our loss function and the Adam optimizer [23] to optimize parameters based on computed gradients. The training process is repeated over multiple epochs (e.g., set to 50), with shuffling and batch division of the data.

**Generation phase**. As shown in Fig. 4, POINTERGUESS generates $K$ guesses for each user using the given old password. Furthermore, we implement the Batch Beam Search algorithm based on [61] to improve efficiency and leverage parallel computing on GPUs. As shown in Fig. 5, the algorithm generates top-$K$ guesses simultaneously for all users (e.g., $M$ users) in a batch, selecting the top candidates based on their probabilities. Password generation process continues until the desired number of guesses is obtained for all users.

**Model hyperparameter configuration**. During the generation process, we perform log-softmax operations on the conditional probability predicted by the model at each timestep. To ensure the predicted probability is not zero, we select $\varepsilon$=1e-12 as our smoothed value. We denote the vocabulary as $\Sigma$, consisting of 95 printable ASCII characters and four special identifiers (i.e., `<EOS>`, `<SOS>`, `<PAD>`, and `<UNK>`), and the vocabulary size $\|\Sigma\|$ is 99. Without loss of generality, we implement Bahdanau et al.'s attention mechanism [7] in our model, and set the learning rate as 0.001 and the number of training epochs as 50. We use the Dropout [48] to alleviate overfitting, and the dropout rate is set to 0.5, which means that there are 50% neurons randomly selected to be invalid and not considered in gradient operations.

## 4 Experiments and analysis

### 4.1 Dataset cleaning and ethical consideration

**Datasets**. We demonstrate the effectiveness of POINTERGUESS and compare it with other state-of-the-art models based on 11 large-scale real-world datasets, a total of 4.8

Table 2: Setups of 14 different attack scenarios (see detailed results in Fig. 6).‡

| #. Attack scenario | Language | Training set setup | Size (pairs) | Testing set setup | Size (pairs) | Clean strategies† |
|---|---|---|---|---|---|---|
| #1. 126 → CSDN | Chinese | 126 → Dodonew | 188,926 | 126 → CSDN | 85,206 | Len≥8 |
| #2. CSDN → 126 | Chinese | CSDN → Dodonew | 211,385 | CSDN → 126 | 86,104 | *Basic* |
| #3. Tianya → CSDN | Chinese | Tianya → Dodonew | 434,255 | Tianya → CSDN | 826,559 | Len≥8 |
| #4. CSDN → Dodonew | Chinese | CSDN → 126 | 86,104 | CSDN → Dodonew | 211,385 | *Basic* |
| #5. 000Webhost → LinkedIn | English | 000Webhost → Yahoo | 265,083 | 000Webhost → LinkedIn | 213,697 | Len≥6 |
| #6. Yahoo → 000Webhost | English | Yahoo → LinkedIn | 40,646 | Yahoo → 000Webhost | 37,479 | `LD` |
| #7. LinkedIn → 000Webhost | English | LinkedIn → Yahoo | 40,812 | LinkedIn → 000Webhost | 259,175 | `LD`, Len≥6 |
| #8. 000Webhost → RedMart | English | 000Webhost → Linkedin | 213,697 | 000Webhost → RedMart | 6,858 | Len≥6 |
| #9. 80% Mixed_EN → 20% Mixed_EN | English | 80% of Mixed_EN | 338,857 | 20% of Mixed_EN | 84,714 | *Basic* |
| #10. 80% Mixed_CN → 20% Mixed_CN | Chinese | 80% of Mixed_CN | 434,255 | 20% of Mixed_CN | 108,564 | *Basic* |
| #11. 80% 4iQ → 20% 4iQ | Mixed | 80% of 4iQ dataset | 116,837,808 | 20 % 4iQ dataset | 29,209,452 | *Basic* |
| #12. 80% COMB → 20% COMB | Mixed | 80% of COMB | 342,921,727 | 20 % COMB dataset | 85,730,432 | *Basic* |
| #13A. Tianya, 126 → Taobao | | Tianya, 126 → Dodonew | | Tianya, 126 → Taobao | | *Basic* |
| #13B. Tianya → Taobao | Chinese | Tianya → Dodonew | 95,457 | Tianya → Taobao | 79,562 | *Basic* |
| #13C. 126 → Taobao | | 126 → Dodonew | | 126 → Taobao | | *Basic* |
| #14A. 80% *Union* → 20% $Union_B$* | | 80% of *Union* dataset | | 20 % $Union_B$ dataset | | *Basic* |
| #14B. 80% $Union_{A1}$ → 20% $Union_B$ | English | 80% of $Union_{A1}$ dataset | 27,833,899 | 20 % $Union_B$ dataset | 10,785,542 | *Basic* |
| #14C. 80% $Union_{A2}$ → 20% $Union_B$ | | 80% of $Union_{A2}$ dataset | | 20 % $Union_B$ dataset | | *Basic* |

‡ $A \rightarrow B$ (e.g., #1. 126 → CSDN) means that: A user's password at service $A$ can be used by an attacker to help attack this user's account at service $B$. Note that for #13A and #14A, we represent the attack scenarios as $A1, A2 \rightarrow B$, which means that a user's passwords at services $A1$ and $A2$ can be used by an attacker to help herself attack the same user's account at service $B$. The sub-scenarios #13B∼#13C and #14B∼#14C are the sub-scenarios for #13A and #14A, respectively.

*The *Union* dataset is built by matching ClixSense, LiveAuctioneers and 4iQ using email. Note that we remove any data item with fewer than three passwords.

†The *Basic* strategy (defined in Sec. 4.1) involves the removal of invalid emails and non-human created passwords. Cleaning strategies need to exclusively focus on the site $B$ in the $A \rightarrow B$ pairs. E.g., for Yahoo → LinkedIn and LinkedIn → Yahoo, the initial number of password pairs is identical. However, after applying the `LD` strategy (which means retaining passwords with at least one letter and one digit) to LinkedIn, its number becomes 40,646, while after applying `LD` and Len≥6 to Yahoo, its number becomes 40,812. This explains the differences in the number of data items for Yahoo → LinkedIn (`LD`) and LinkedIn → Yahoo (`LD` and Len≥6).

billion passwords (see Table 1). To ensure a comprehensive and reliable evaluation of our models and their counterparts, besides four English and five Chinese datasets, we further employ two synthesised large-scale datasets, 4iQ [2] and COMB [3]. All these datasets in Table 1 were leaked and published on the Internet from 2011 to 2021 and are representative of current real users' reuse behavior. Particularly, (1) 4iQ and COMB are two massively mixed datasets [2,3] consisting of large datasets exposed by previous breaches (e.g., LinkedIn, Netflix); (2) RedMart [4] is an online grocery platform whose servers store all passwords in salted-hash. These passwords serve as *real* targets (as with [57]); (3) 000Webhost is a website used by web administrators, so its users are more likely to have higher security awareness. Thus, the experiments involving 000Webhost in attack scenarios #6 and #7 represent evaluation on security-savvy users (see Table 2).

**Ethical consideration**. Though ever available on the Internet and dark web (and widely used in the literature [14,38,57,58]), these datasets are private data. Thus, we take three precautionary measures to ensure that there is no additional harm to users: (1) All our datasets are stored and processed on computers not connected to the Internet; (2) Only report the aggregated statistical information and some typical password examples (without PII), i.e., treat each account as confidential, so that using it in our work will not bring new risks to the corresponding victim; (3) Delete all the intermediate sensitive data (e.g., email and datasets of target guesses) once our analysis is completed. These datasets may be exploited by malicious parties for misconduct, while our use is beneficial for the community to understand password strength and for security administrators to make more informed decisions.

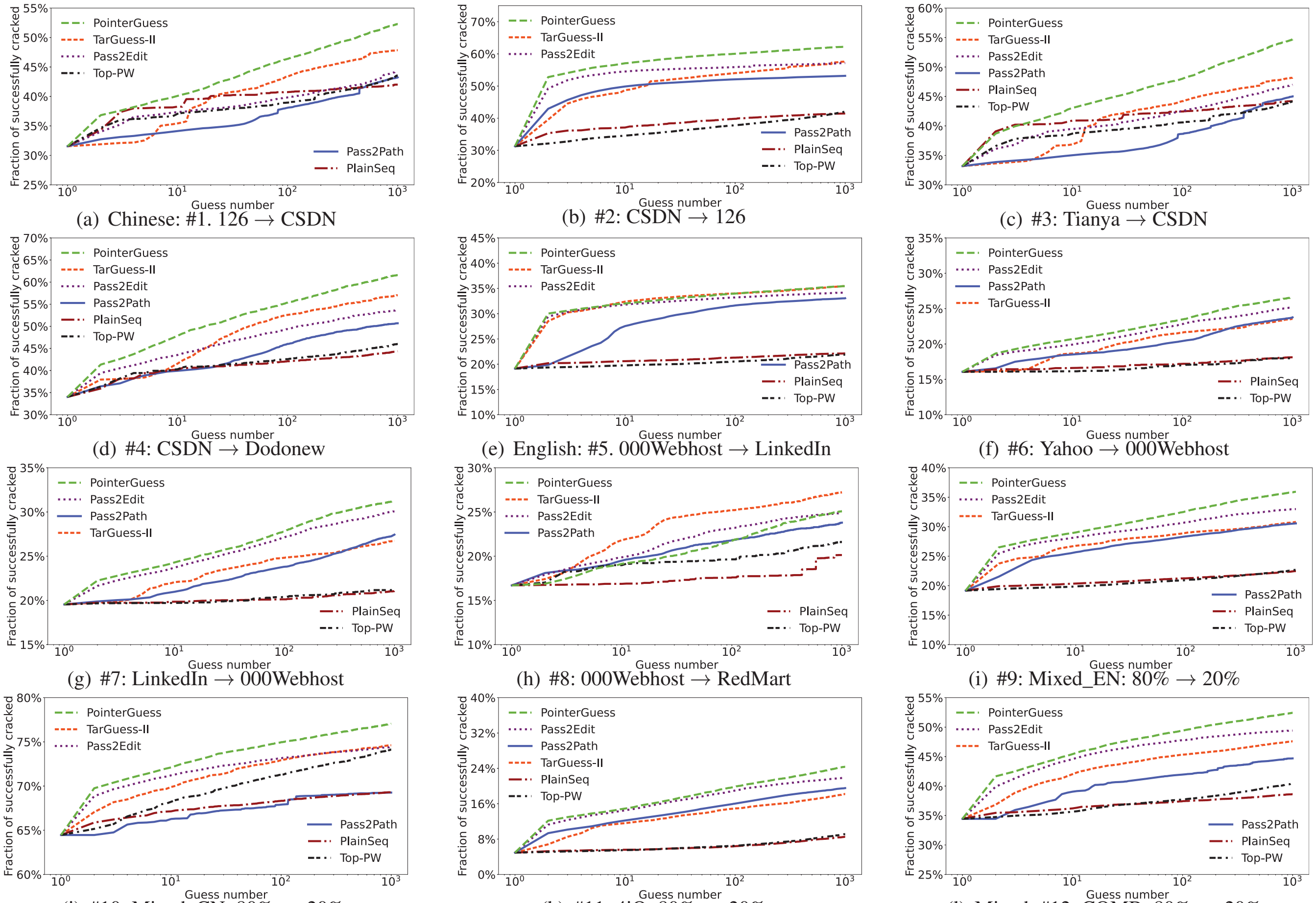
**Dataset cleaning**. First, we remove data items with an empty/invalid email. We also keep passwords that are *Len*<30 and only contain 95 printable ASCII characters. We call this as the *Basic* cleaning strategy. We will adaptively adjust this strategy for different websites according to their password policy (see more details in Table 1).

## 4.2 Experimental setup

To better evaluate the effectiveness of our PointerGuess, we design 12 practical attack scenarios (see Table 2) by employing datasets described above (see Table 1). More specifically, we use email to match two datasets to create training/test datasets. For instance, 126 → CSDN means matching 126 with CSDN (by using email) and getting 85,206 password pairs. We design attack scenarios #1∼#4 and #5∼#8 for Chinese and English users, respectively. All selected test sets, except for the scenario #8 (where RedMart passwords are in salted-hash), are in plaintext. As 000webhost is mainly used by web administrators, and we design attack scenarios #6 and #7 to simulate attacks on high-security users.

Scenarios #9 and #10 employ mixed datasets within one language, i.e., Mixed_EN and Mixed_CN, which combine multiple English (i.e., 000Webhost, LinkedIn and Yahoo) and Chinese (i.e., Tianya, Dodonew, and CSDN) datasets, respectively. Scenarios #11 and #12 further evaluate users of mixed languages by employing two large-scale synthesized datasets 4iQ and COMB. These four additional evaluation setups are in accord with that of [57].

In Sec. 6, we introduce MS-PointerGuess designed for multiple password reuse attack scenarios. To demonstrate its effectiveness, we design two practical attack scenarios #13 and #14 (see Table 2 for details), with each consisting of a main scenario for MS-PointerGuess and two sub-scenarios for comparison with PointerGuess. We design scenario #13 (#13A∼#13C) for Chinese users, which evaluate the cases when the attacker gets *two* old passwords of the victim. This allows us to fairly compare the effectiveness of




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

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

(g) #7: LinkedIn → 000Webhost (h) #8: 000Webhost → RedMart (i) #9: Mixed_EN: 80% → 20%

(j) #10: Mixed_CN: 80% → 20% (k) #11: 4iQ: 80% → 20% (l) Mixed: #12. COMB: 80% → 20%

Figure 6: Experiments for attack scenarios #1∼#12, for each of which the training set is shown in Table 2 and the test set is as the sub-title. Our POINTERGUESS achieves the highest cracking success rate in 10 of 12 scenarios over its foremost counterparts TarGuess-II [56], Pass2Edit [57] and Pass2Path [38].

POINTERGUESS with MS-POINTERGUESS.

We design scenario #14 (#14A∼#14C) for English users. We match ClixSense, LiveAuctioneers, and 4iQ by emails, ensuring a minimum of three passwords for each data item in the matched dataset called $Union$. We randomly select 80% of the $Union$ as the training set and the rest 20% as the test set. Additionally, to comprehensively evaluate our models' performance, we construct $Union_{A1}$ and $Union_{A2}$ for scenarios #14B and #14C, based on the source data, respectively.

**State-of-the-art models for comparison**. We compare our model with three state-of-the-art models (i.e., TarGuess-II [56], Pass2Path [38], and Pass2Edit [57] and their variants), as well as other relevant models (e.g., Top-PW and PlainSeq). To evaluate the impact of the pointer module on model performance, we use a basic seq2seq model, called PlainSeq, *without* utilizing the pointer module. We provide a briefly overview of these models in Appendix A. As CPG [40] and ReSeg-PCFG [51] are tailored for trawling guessing or mask guessing rather than password reuse-based attacks, and RFGuess-reuse achieves comparable performance with TarGuess-II [56] and Pass2Path [38] (see Table 5 in [58] for details), we exclude them from our model comparison.

**Experimental environment**. We randomly sample 20,000 passwords for test sets exceeding 20,000 as previous studies [38,57] have proved that using over 10,000 password pairs leads to performance convergence. We perform all experiments on a workstation with an Intel Xeon Silver processor and a GPU of NVIDIA RTX 3090 (24GB of VRAM), an experimental environment most attackers can easily build.

## 4.3 Experimental results

Here we briefly analyze the results of attacking scenarios #1∼#12. As with [14, 38, 57, 58], we use the guess-number graph to evaluate the performance of POINTERGUESS and its counterparts (see more details about these models in Appendix A). Moreover, we present exact crack rates for specific guess numbers (e.g., 10, 100, and 1,000), which are commonly concerned in password security studies [38, 56, 57] and standard (e.g., NIST [19]). See more detailed results in Appendix G of the full version at https://bit.ly/3wGx9Ke.

**Overall analysis**. Due to the presence of identical password pairs (i.e., $pw_A = pw_B$) in each test set, we present the experimental results of scenarios #1∼#12 in two cases: one without identical password pairs and the other with.

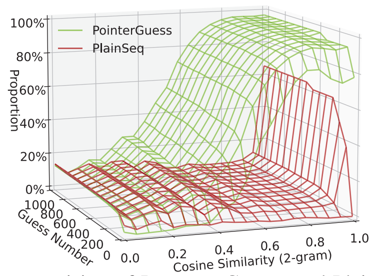



Figure 7: The comparision of POINTERGUESS and PlainSeq's ability in cracking passwords with different cosine similarity (2-gram) thresholds. Here we take scenario #2 (CSDN → 126) for example.

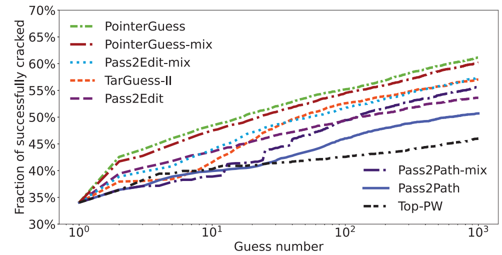



Figure 8: The cracking success rate of all models with mixing popular passwords (using scenario #4: CSDN → Dodonew as an example). A model with the suffix "-mix" means that it is an adjustment to its original model by mixing popular passwords in the same strategy with [57].

In the former case (i.e., without identical password pairs), as shown in Table 8 of the full version, within 1,000 guesses, the success rates of POINTERGUESS are 10.08% ∼ 45.00% (avg. 25.85%), while that of Pass2Edit [57], Pass2Path [38], and TarGuess-II [56] are 9.82%∼37.62% (avg. 20.39%), 8.52%∼31.87% (avg. 16.28%), and 8.92%∼38.38% (avg. 20.61%). That is, the guessing success rates of POINTERGUESS are 21.58%, 52.27%, and 20.61% (on average) higher than Pass2Edit, Pass2Path, and TarGuess-II, respectively.

In the latter case (i.e., with identical password pairs), as shown in Table 9 of the full version, within 1,000 guesses, the success rates of POINTERGUESS are 24.36% ∼ 77.03% (on average 44.91%), while that of Pass2Edit [57], Pass2Path [38], and TarGuess-II [56] are 21.88%∼74.39% (on average 41.25%), 19.54%∼69.26% (on average 38.64%), and 18.20%∼74.62% (on average 41.25%), respectively. That is, the guessing success rates of POINTERGUESS are 8.87%, 16.23%, and 8.87% (on average) higher than Pass2Edit, Pass2Path, and TarGuess-II, respectively.

**Compare with the baseline (PlainSeq)**. To better illustrate the role of the pointer mechanism [50], we conduct a further comparison between POINTERGUESS and the basic seq2seq model [49] without the pointer mechanism [50], i.e., PlainSeq. Fig 7 shows their performance in cracking passwords under different cosine similarity (2-gram) thresholds. Results show that POINTERGUESS drastically outperforms PlainSeq, especially in cases where the cosine similarity threshold ranges from 0.8 to 1.0. We further analyze the password cracked by POINTERGUESS, and find that POINTERGUESS excels in cracking: 1) password pairs whose target passwords are created by adding or deleting uncommon substrings from the old passwords (e.g., `585129wupan` → `585129`); and 2) password pairs whose cosine similarities are large but also with a large edit distance, such as `1000020000` → `100200` whose cosine similarity is 0.91 and edit distance is 4.

**The impact of language on performance**. We now compare the effectiveness of POINTERGUESS in Chinese (#1∼#4) and English (#5∼#8) attack scenarios. As shown in Fig. 6, the experimental results demonstrate that POINTERGUESS outperforms other models in all Chinese scenarios. In English scenarios, our model still achieves higher or comparable performance. Notably, POINTERGUESS significantly outperforms other models when attacking security-savvy users (see scenarios #6 and #7 that attack 000webhost).

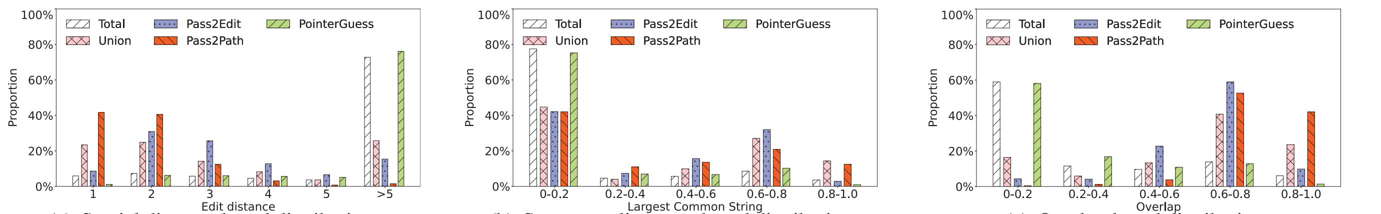
It is worth noting that our POINTERGUESS shows a much higher success rates in Chinese scenarios than in English ones. This can be largely attributed to the facts that: (1) English scenarios all involve users of 000Webhost, who are web administrators and thus possess a higher level of security awareness than common users [56]; (2) there exist vast differences in structural and semantic characteristics between Chinese and English passwords, and the strength of Chinese passwords is weaker in online guessing scenarios (i.e., when the guess number allowed for the attacker is small [55]).

**Mixing with external popular passwords**. We explore the impact of mixing external popular passwords on the performance of POINTERGUESS and other models in attack scenarios #1∼#12. We adopt the same mixing strategy as in [57]. Fig. 8 shows that all models (except for POINTERGUESS) have a significant increase in crack rates after mixing external popular passwords. Pass2Edit [57] and Pass2Path [38], in particular, improve performance significantly after mixing popular passwords, which is mainly due to their inherent defect of overlooking users' macroscopic population-wide reuse behaviors (see Sec. 1.1). While excluding identical password pairs, within 1,000 guesses, the success rates of POINTERGUESS-mix are 10.38%∼45.50% (avg. 25.87%), while that of Pass2Edit-mix, Pass2Path-mix are 10.96%∼45.70% (avg. 24.58%) and 8.52%∼31.87% (avg. 16.28%), respectively. That is, the guessing success rates of POINTERGUESS-mix are 5.25% and 58.91% (on average) higher than Pass2Edit-mix and Pass2Path-mix, respectively.

While not excluding identical password pairs, within 1,000 guesses, the success rates of POINTERGUESS-mix are 23.95%∼62.55% (avg. 44.94%), while that of Pass2Edit-mix, Pass2Path-mix are 22.96%∼62.68% (avg. 44.04%) and 19.85%∼57.97% (avg. 41.27%). That is, the guessing success rates of POINTERGUESS-mix are 2.04% and 8.89% (on average) higher than Pass2Edit-mix and Pass2Path-mix.

## 5 Further analysis

We analyze characteristics of passwords cracked by different models, employing various similarity metrics to explore users' password reuse from both "syntactic" and "semantic" perspectives. To demonstrate models' performance accurately, we adopt a new "password reuse" definition (as discussed in




(a) Spatial distance-based distribution (b) Sequence alignment-based distribution (c) Overlap-based distribution

Figure 9: Syntactic metric distributions of independently cracked password pairs by three major models. Figs. 9(a)∼9(c) show the example of similarity distributions based on spatial-distance metric (i.e., edit distance [27]), sequence-alignment metric (i.e., Largest Common String algorithm [14]) and overlap-based metric (i.e., Overlap [26]), respectively. "Total" represents all password pairs in all test sets and "Union" represents all password pairs cracked by three models.

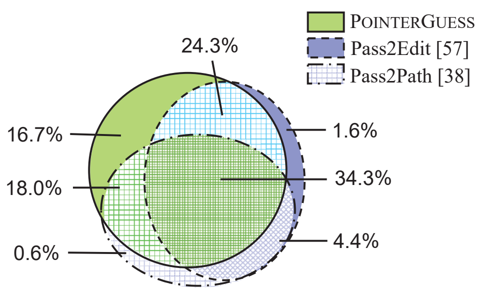



Figure 10: The overlap ratios of cracked password pairs by three models.

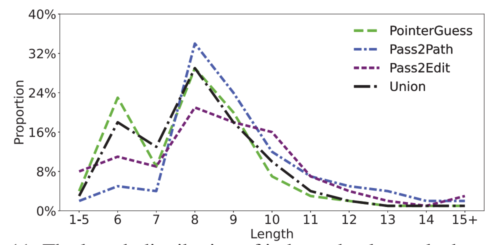



Figure 11: The length distribution of independently cracked passwords.

Sec. 3.1) to measure similarity distributions. Additionally, we investigate existing models' performance and limitations based on similarity distributions of cracked passwords.

## 5.1 Characteristics of cracked passwords

**Overlap**. To compare the cracking capabilities of POINTERGUESS, Pass2Edit [57], and Pass2Path [38], we examine the overlap in *uniquely* cracked password pairs across 12 attack scenarios (i.e., 29,252 of 89,951 all cracked password pairs). Fig. 10 shows a total overlap rate of 34.3% (10,033 of 29,252) among the three models. For *independently* cracked password pairs, POINTERGUESS has a 16.7% overlap (4,885 of 29,252), while that of Pass2Edit and Pass2Path are 1.6% (468 of 29,252) and 0.6% (176 of 29,252), respectively.

Notably,the overlap between POINTERGUESS and either Pass2Edit or Pass2Path is significantly higher (i.e., 24.3% = 7,108/29,252, and 18.0% = 5,265/29,252, respectively) compared to the intersection between Pass2Edit and Pass2Path (i.e., 4.4% = 1,287/29,252). This highlights that POINTERGUESS is good at what Pass2Edit and Pass2Path can do.

**Length distribution**. Here we use the password pairs *independently* cracked by each of these three models to explore their differences in cracking passwords of varied lengths. Fig. 11 shows that our model primarily focuses on passwords with lengths of 6 and 8∼10 due to the fact that POINTERGUESS can capture users' macroscopic population-wide reuse behaviors, e.g., using popular passwords (which are typically of length 6 and 8) and substituting long segments (like `yjqqq916198` → `916198yj`). Notably, the length distributions of our model and the union set (denoted as "*Union*" in Fig. 11) exhibit a high degree of similarity (i.e., have two peaks), which highlights that POINTERGUESS is good at cracking passwords of lengths that Pass2Edit [57] *and/or* Pass2Path [38] are good at.

## 5.2 Characterize password reuse behaviors

We conduct further evaluations on the performance of different models in characterizing user's password reuse behaviors. We employ various similarity metrics (as we mentioned in Sec. 2.2) to evaluate different models' ability on cracking password pairs with different similarity scores.

First, we employ syntactic metrics to measure the distribution of independently cracked password pairs. As shown in Fig. 9, POINTERGUESS exhibits superior performance on low-similarity cases, *attributing this to its ability in predicting targeted password character sequences without excluding unsimilar password pairs from the training set.* This allows POINTERGUESS to generate tweaked passwords with low similarity to the old password, including those with edit distances >5 (see examples in Table 3).

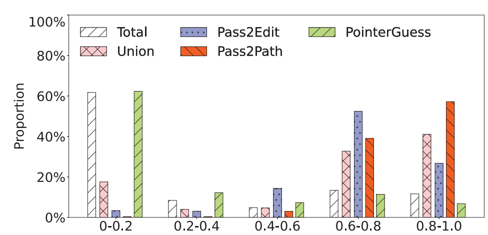



Figure 12: The combined similarity distribution of edit distance and cosine similarity (2-gram) as defined in [20]. Results are similar to Fig. 9.

Fig. 9 illustrates that Pass2Edit [57] and Pass2Path [38] exhibit "overfitting", as the similarity distribution of the cracked password pairs significantly *deviates* from the overall similarity distribution (i.e., "Total" in Fig. 9) of all password pairs in the test sets. This phenomenon implies that Pass2Edit and Pass2Path struggle to model the distribution of entire password reuse behaviors. Particularly, they face challenges in cracking password pairs with excessively long

Table 3: Ten examples of password pairs cracked independently by POINTERGUESS, Pass2Edit [57] and Pass2Path [38].

| Models | POINTERGUESS | | Pass2Edit [57] | | Pass2Path [38] | |
|---|---|---|---|---|---|---|
| Index | Old password | Target password | Old password | Target password | Old password | Target password |
| 1 | 852255685145294 | abc123 | MCfaraona020591 | mcfaraona91 | 8841800lin | lin8841800lin |
| 2 | boy78697740 | boy123456789 | edwardcullenqwe | Edwardcullen | jangobango88 | jangobango1988 |
| 3 | kazeevatanyuffka872ghbrjyf | kazeevatanyuffka | Castor | Castor08 | 13197277038 | 131w97277038 |
| 4 | katmarlzelda969 | katmarlzelda969@yahoo.com | 4.14495E | 4.14495E+13 | IloveYOU2998 | iloveyou2998 |
| 5 | ghostgamer-2001 | ghostgamer-2001@hotmail.com | t0romerda. | toromerda | SAIIIOK | sailiok |
| 6 | uuDBUMDM5NApOzYW | qweasdzxc | UHJVuhjvbr49 | Uhjvuhjvbr49 | wgpfuqd861208 | wgpfUQD861208 |
| 7 | jaydiltddasilva@partners.org | jaydilla1 | 30061986123 | 30061986qwe | rajuraju | raju2raju |
| 8 | 102457685& | 102457685!! | WMOOLMAN1058 | WMOOLMAN | drdeath | 1DRDEATH |
| 9 | 1991322322 | 1.99132E+12 | RBV//1960 | rbl//1960 | samantha | s@mantha |
| 10 | 6125251987110 | 6.12525E+12 | SharmaHellV1.0 | HellV1.0 | liljojo202 | liljojo120 |

Table 4: The proportion of transformation rules.

| Models | Password transformation rules %† | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Sub. | Leet | Cap. | LCS | Rev. | Leet+LCS | Cap.+Sub. | Others |
| Union | 71.48 | 0.57 | 4.31 | 63.71 | 0.06 | 36.48 | 1.78 | 9.78 |
| POINTERGUESS | 62.46 | 0.53 | 3.35 | 57.41 | 0.13 | 34.94 | 5.00 | 20.17 |
| Pass2Edit [57] | 76.75 | 0.60 | 4.53 | 68.03 | 0.01 | 38.69 | 7.02 | 3.13 |
| Pass2Path [38] | 78.92 | 0.59 | 5.53 | 68.14 | 0.00 | 36.15 | 8.44 | 1.92 |

†The abbreviations "Sub.", "Cap.", "LCS" and "Rev." stand for "Substring", "Capitalization", "Largest Common String" and "Reversal" respectively.

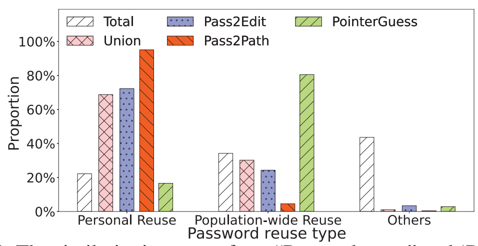
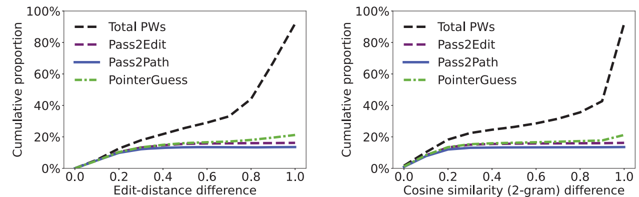


Figure 13: The similarity in terms of our "Personal reuse" and 'Population-wide reuse' and other types of reuse.

editing sequences or low similarity (e.g., KeveinMobile → password123). As shown in Fig. 12, we combine edit distance and cosine similarity (2-gram) as suggested in [20] for a comprehensive evaluation, yielding similar conclusions to other syntactic metrics.

Second, we employ semantic metrics, following the workflow proposed by Wang et al. [52], to measure the transformation rules of cracked password pairs. Table 4 shows that Pass2Edit [57] and Pass2Path [38] tend to focus on some specific transformation (e.g., "LCS") while overlooking some other complex segment-level transformations (i.e., "Others" in Table 4), e.g., transforming yjqqq916198 to 916198yj.

Third, as mentioned in Sec. 3.1, we introduce a hierarchical definition of "password reuse", and propose a workflow that considers both syntactic and semantic metrics (see Fig. 2). More specifically, we use a combination of edit distance and cosine similarity to identify "personal reuse" password pairs, while applying transformation rules to detect "population-wide reuse" patterns for the remaining password pairs. As shown in Fig. 13, nearly 40% of all test passwords are categorized as population-wide reuse, and POINTERGUESS shows a clear advantage in cracking such password pairs. We can see that another 40% of password pairs still can not be identified as reuse (see "Total" when x-axis="Others") by POINTERGUESS and other major models. This outlines the need for a more thorough understanding of users' reuse behaviors.



(a) Edit distance. (b) Cosine similarity.

Figure 14: Cumulative Distribution Functions (CDF) of cracking proportion on password similarity difference. Figs. 14(a) and 14(b) show the results of using edit distance and cosine similarity as metrics, respectively.

Table 5: Running time of different attack models.†

| Attack model | POINTERGUESS | Pass2Edit [57] | Pass2Path [38] |
|---|---|---|---|
| Training time | 15:14 | **09:43** | 14:10 |
| Testing time | **00:24** | 02:26 | 01:47 |
| Speed‡ (pw/s) | **9,700∼9,800** | 2,100∼2,200 | 2,900∼3,000 |
| Model size (MB) | **2.26** | 11 | 53.6 |

†All running time is taken from attack scenario #10, and their format is "hour:minute". All model parameters are consistent with Sec. 3.

‡Speed (of generating guesses) is calculated by dividing the total number by the total testing time, and there may be fluctuations in different scenarios.

## 5.3 Further exploration on model performance

**Limitation of existing models**. Existing models tend to lean towards generating passwords that are either "very similar" or involve "fewer edit operations" regarding old passwords. This limits their ability to effectively comprehend and capture password reuse behaviors across the entire password distribution, particularly in cases of population-wide reuse. As a result, there is a notable gap between their cracking performance and the reality of password reuse.

To demonstrate this disparity and delve deeper into the limitations of password reuse-based guessing models, we utilize Cumulative Distribution Function (CDF) curves. As shown in Fig. 14, existing models [38, 57] (that focus on generating highly similar passwords) quickly reach the stable saturation plain. In contrast, POINTERGUESS can crack password pairs even when the similarity differences are as large as 0.8∼1.0, which corroborates its capability in capturing "population-wide reuse" and provides a new perspective on why POINTERGUESS achieves higher performance than existing models.

Furthermore, when comparing the proportion cracked by the three models with the whole 89,951 unique test password pairs, a significant gap emerges. As shown in Fig. 14, the CDF curves of proportions cracked by three models deviates significantly from that of overall password pairs when the similarity difference $\geq 0.5$. To address this issue, we intro-

Table 6: Comparison of the cracking success rate of MS-POINTERGUESS and POINTERGUESS.*

| Experiment setup | | Include the identical password pairs | | | Remove the identical password pairs | | |
|---|---|---|---|---|---|---|---|
| Attack scenario | Guess number | POINTERGUESS-A1&A2 | POINTERGUESS-A1 | POINTERGUESS-A2 | POINTERGUESS-A1&A2 | POINTERGUESS-A1 | POINTERGUESS-A2 |
| #13: Tianya, 126 $\rightarrow$ Taobao | 10 | **43.44%** | 32.22% | 41.41% | **2.70%** | 2.35% | 2.14% |
| | 100 | **44.82%** | 34.18% | 43.12% | **5.08%** | 4.29% | 4.38% |
| | 1,000 | **45.97%** | 35.62% | 44.45% | **7.05%** | 5.79% | 6.36% |
| #14: 80%$Union \rightarrow$ 20%$Union$ | 10 | **58.97%** | 47.90% | 44.30% | **10.34%** | 6.07% | 9.54% |
| | 100 | **61.37%** | 49.70% | 47.67% | **15.58%** | 8.97% | 15.00% |
| | 1,000 | **63.67%** | 52.11% | 50.80% | **20.63%** | 12.99% | 20.32% |

*Note that the POINTERGUESS-A1&A2 represents our MS-POINTERGUESS. POINTERGUESS-A1 means that we feed $pw_{A1}$ into POINTERGUESS and guess $pw_B$, and similarly for the definition of POINTERGUESS-A2. POINTERGUESS-A1&A2 means that we feed both the $pw_{A1}$ and $pw_{A2}$ into the MS-POINTERGUESS and guess $pw_B$.

duce our MS-POINTERGUESS for multiple leaked password scenarios (see details in Sec. 6).

**Model attacking efficiency**. Here we examine the attacking efficiency of different neural network-based models. Fig. 5 shows that the training time of POINTERGUESS is slightly longer than Pass2Edit [57] and Pass2Path [38], mainly because existing models need to filter the training set with some similarity threshold (e.g., edit distance<4) and this leads to a smaller training set; As for testing time, our POINTERGUESS runs the fastest generation speed, which is much higher than other models; Our model's size is only 2.26 MB, which is 5∼25 times smaller than other models, leading to easier local deployment and reducing the risk of information leakage.

**Model preferences**. Table 3 shows ten examples of password pairs cracked independently by different models. Our POINTERGUESS demonstrates clear preferences in complex reuse behaviors. First, POINTERGUESS can characterize users' vulnerable behaviors of reusing popular passwords (e.g., `abc123` in the first example). Second, POINTERGUESS can generate semantic fragments based on the old password (e.g., email suffixes like `@hotmail.com`), potentially assisting in guessing reused passwords. Third, POINTERGUESS utilizes extractive generation by identifying key parts of the old password to generate the target password, (see indexes 7∼10 in Table 3). Particularly, our model can generate numbers represented in scientific notation (e.g., `1991322322` → `1.99132E+12`). This reflects the limitations in Pass2Edit [57] and Pass2Path [38], which utilize the old password as a template for transformations (e.g., insert/delete a character) is less effective in identifying reused fragments within the old password.

**Potential applications in password protection**. We consider two potential applications of POINTERGUESS: password strength meter (PSM) and compromised credential checking (C3) services. We discuss how to design reuse-based PSM in Sec. 7 and the application in C3 services in Appendix B.

# 6 Multi-Source PointerGuess

Reports [13, 31] show that most users maintain over 90 accounts on average and prefer to reuse their old passwords, which impairs password security. At IEEE S&P'19, Pal et al. [38] attempted to employ multiple old passwords of a user for targeted guessing attacks by running Pass2Path multiple times and simply "merge the lists by picking one from each list in a round robin manner" [38]. Their ad hoc approach cannot accurately capture the relationships between different old passwords, and we take a principled approach.

## 6.1 Modeling password generation

In response to the threat posed by a realistic attack scenario involving multiple leaked old passwords, we introduce a new targeted password guessing model, MS-POINTERGUESS. MS-POINTERGUESS incorporates a "Multi-Encoder" module into POINTERGUESS to handle the victim's multiple leaked passwords simultaneously. More specifically, this module empowers the model to extract multiple contexts from each encoder input (i.e., old password), enabling our MS-POINTERGUESS to effectively characterize users' password reuse behaviors by leveraging multiple contexts.

Note that MS-POINTERGUESS is highly scalable and can be easily extended to handle multiple old passwords. Here, we use two encoders to briefly describe MS-POINTERGUESS without losing generality.

In general, the conditional guessing probability that an attacker exploits users' multiple leaked passwords (e.g., $pw_A$ at site $A$ and $pw_B$ at site $B$) to attack the victim's target password at site $C$ (namely $pw_C = \left(c_1^{'}, ..., c_M^{'}\right)$) can be expressed similarly to Eq. 1, which is

$$P\left(pw_C|pw_A, pw_B\right) = \prod_{i=1}^{M} P\left(c_i^{'}|c_{<i}^{'}, pw_A, pw_B\right), \tag{8}$$

where $c_{<i}^{'}$ dentoes the sub-sequence $\left(c_0^{'}, ..., c_{i-1}^{'}\right)$ of $pw_C$, and $P\left(c_i^{'}|c_{<i}^{'}, pw_A, pw_B\right)$ is the conditional probability of generating the character $c_i^{'}$ at position $i$ in $pw_C$ given the sub-sequence $c_{<i}^{'}$ and the two old passwords $pw_A$ and $pw_B$.

There are two key research questions (**RQs**) that need to be solved to build an effective targeted password guessing model based on users' multiple leaked passwords:

**RQ1**: How to evaluate the importance of different leaked passwords on guessing the target password?

**RQ2**: Does the designed model demonstrate superior performance compared to POINTERGUESS in practical?

To address **RQ1**, we introduce an *additional softgate layer*, denoted as $\lambda$, which serves to evaluate the significance of users' various old passwords and discern their preferences when generating a new password. More specifically, MS-POINTERGUESS outputs two context vectors, $c_t, c_t^{'}$, from $pw_A$

and $pw_B$, respectively. Then, we can express $\lambda \in [0,1]$ as

$$\lambda = \sigma\left(W_c * c_t + W_{c'} * c_t^{'} + b_\lambda\right), \tag{9}$$

where the $\sigma$ is a sigmoid function, $W_c$, $W_{c'}$, $b_\lambda$ are learnable parameters. Initially, $\lambda$ is employed to weigh and combine the conditional probabilities of copying characters from $pw_A$ (i.e., $P^A_{copy}$) and $pw_B$ (i.e., $P^B_{copy}$), respectively. This yields $P_{copy}$, the weighted conditional probability of copying characters from $pw_A$ and/or $pw_B$, can be expressed as:

$$P_{copy}(c) = \lambda * P^A_{copy}(c) + (1-\lambda) * P^B_{copy}(c). \tag{10}$$

Then, we can represent $P^A_{copy}$ and $P^B_{copy}$ similarly to Eq. 4, which are

$$P^A_{copy}(c) = FFN\left(\sum_{i:c_i=c} \alpha^t_i\right), \tag{11}$$

and

$$P^B_{copy}(c) = FFN\left(\sum_{i:c_i^{'}=c} \alpha'^t_i\right), \tag{12}$$

where $\alpha^t_i$ and $\alpha'^t_j$ are the attention weights of the two encoders at timestep $t$ and $c_i$ (resp. $c_i^{'}$) denotes the character at position $i$ in $pw_A$ (reps. $pw_B$). $FFN(\cdot)$ is a feed-forward network. Note that if character $c$ does not appear in $pw_A$ or $pw_B$, then the value of $\sum_{i:c_i=c} \alpha^t_i$ or $\sum_{i:c_i^{'}=c} \alpha'^t_i$ will be zero.

Still, we use the pointer mechanism $p_g$ to weigh $P_{copy}(c)$ and $P_{vocab}(c)$, the conditional probability of generating characters from the vocabulary. At each timestep $t$, our generation probability $p_g \in [0,1]$ can be calculated from two content vectors and current decoder state vector $s_t$ and current decoder input $x_t$, that is

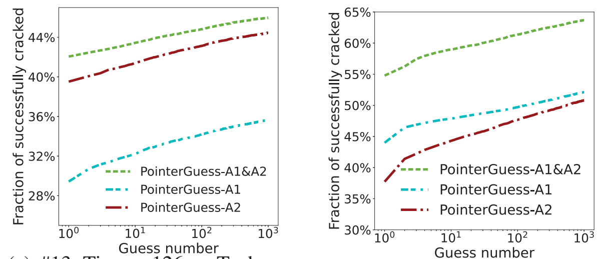
$$p_g = \sigma\left(W_c * c_t + W_{c'} * c'_t + W_s * s_t + W_x * x_t + b_g\right), \tag{13}$$

where $W_c$, $W_{c'}$, $W_s$, $W_x$, $b_g$ are learnable parameters, and $\sigma(\cdot)$ is a sigmoid function.

Finally, MS-PointerGuess integrates $P_{copy}$ and $P_{vocab}$ to represent $P_{gen}(c)$, the conditional guessing probability of MS-PointerGuess generating the character $c$, which is

$$P_{gen}(c) = p_g * P_{copy}(c) + (1-p_g) * P_{vocab}(c). \tag{14}$$

Overall, the scalability facilitated by the "Multi-Encoder" module in MS-PointerGuess empowers our model to handle users' multiple leaked passwords simultaneously. This capability enables MS-PointerGuess to extract multiple contexts from various old passwords, facilitating flexible decisions regarding the importance of each old password at every timestep. Moreover, the pointer mechanism [50] ensures that our MS-PointerGuess dynamically determines whether to copy characters from the old passwords or generate new characters directly from the vocabulary.




(a) #13: Tianya, 126 → Taobao (b) #14: 80% *Union* → 20% *Union*

Figure 15: Experiments of attack scenarios for MS-PointerGuess (i.e., #13 and #14) in Table 2. PointerGuess-A1&A2=MS-PointerGuess.

## 6.2 Experimental results and analysis

**Experimental design and results**. To address **RQ2**, we conduct two practical attack scenarios (#13 and #14, detailed in Sec. 4.2) to evaluate the performance of MS-PointerGuess. Fig. 15 shows that MS-PointerGuess invariably outperforms PointerGuess across both attack scenarios. For identical password pairs, MS-PointerGuess achieves cracking success rates that are, on average, 17.54% (scenarios #13) and 26.11% (scenarios #14) higher than PointerGuess within 100 guesses, respectively. Even when excluding identical password pairs, MS-PointerGuess maintains its superiority. More specifically, within 100 guesses, it achieves cracking success rates that are, on average, 17.20% (scenarios #13) and 38.78% (scenarios #14) higher than PointerGuess, respectively. See Table 6 for more details and specific results.

This highlights the effectiveness of MS-PointerGuess and the substantial impacts of utilizing different training sets to attack the same test set on PointerGuess' efficiency. As a large proportion of users directly reuse their old passwords (i.e., 20%∼59% [14, 51, 53, 57]) and there are unending catastrophic password leaks [44, 62] (making it more and more likely that users have leaked two or more distinct passwords), password guessing based on multiple old passwords is a rather damaging threat (see the columns 3 and 6 in Table 6).

**Further analysis**. Overall, our analysis reveals two key findings: (1) The results show that most users have identical old passwords, aligning with recent research findings [13, 18]. As users' leaked passwords increase, the risk of compromising their target passwords also rises. Directly using these identical password pairs in password cracking significantly improves the success rate; (2) MS-PointerGuess effectively leverages multiple old passwords to generate accurate guesses, resulting in higher cracking rates compared to PointerGuess.

# 7 Targeted Password Strength Meters

Password strength meters (PSMs) offer real-time feedback on password security during user registration, receiving much attention as a useful tool [54]. Among them is the Zxcvbn [60], a widely-used PSM that is renowned for its accuracy, low cost, and user-friendliness. While Zxcvbn performs well under trawling guessing attacks, it does not consider the targeted guessing threat as explored in the previous sections.

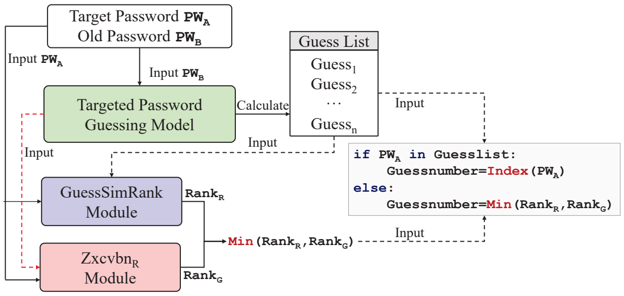



Figure 16: Architecture of PR-PSM. It consists of three primary parts: (1) search the index in the guess list; (2) the *Zxcvbn*$_R$ module; (3) the *GuessSimRank* module.

To address this limitation, we introduce PR-PSM, a password reuse-based PSM that integrates our POINTERGUESS with Zxcvbn. As shown in Fig. 16, PR-PSM utilizes users' old passwords to accurately estimate the password strength through a "multi-step evaluation" process. This process includes *GuessSimRank* and $Zxcvbn_R$ modules, to accurately evaluate the password strength of the target password.

## 7.1 Multi-step evaluation

Fig. 16 shows the architecture of our PR-PSM. We design a two-step evaluation mechanism to evaluate the strength of a given password more accurately.

**Step 1: Get the index of the target in the guess list**. First, we directly input the old password $pw_B$ into the targeted password guessing model (e.g., POINTERGUESS) and generate Top-$K$ guesses. Then, if $pw_A$ is in the guess list, we use its *index* in the guess list as the guess number, that is

$$GN = Index(pw_A), \tag{15}$$

where $GN$ denotes the guess number (i.e., the *index* of $pw_A$). If $pw_A$ is not in the guess list, we move into the next step.

**Step 2: Get the guess number from two modules**. When $pw_A$ is not in the guess list, PR-PSM uses two evaluation modules to assess the strength of the target password. The first module, $Zxcvbn_R$, integrates Zxcvbn [60] and evaluates the strength $Rank_R$ of $pw_A$ using the generated guess list, which can be expressed as

$$Rank_R = Zxcvbn_R\left(pw_A\right). \tag{16}$$

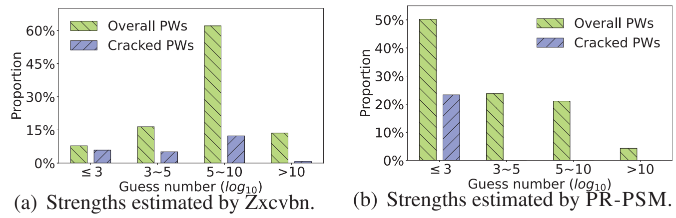
The second module, *GuessSimRank*, integrates POINTERGUESS and Zxcvbn to evaluate the strength $Rank_G$ of $pw_A$, that is

$$Rank_G = GuessSimRank\left(pw_A\right). \tag{17}$$

The final guess number $GN$ is determined as the minimum between $Rank_R$ and $Rank_G$, that is

$$GN = min(Rank_R, Rank_G). \tag{18}$$

Alg. 1 shows the detailed design of PR-PSM. Then we describe how we use $Zxcvbn_R$ and *GuessSimRank* two modules to reevaluate the strength of the target password $pw_A$.




(a) Strengths estimated by Zxcvbn. (b) Strengths estimated by PR-PSM.

Figure 17: PSM accuracy evaluation by comparing the distribution of cracked passwords with overall test passwords. The smaller the guess number, the weaker the password, and should the higher probability to be cracked. Results show the advantages of PR-PSM in evaluating password strength with users' old passwords over Zxcvbn [60].

**Algorithm 1:** PR-PSM evaluation

**Data:** Target password, Targeted password guessing model, $K$, Index importance distribution
**Result:** Guess Number $GN$

```
pw_A ← Target password
Model ← Targeted password guessing model /* input
    the password and guess number, output guesses. */
Guesses ← Model(pw_A, K) /* get the guess list,
    which has K guesses. */
Index ← Search(pw_A, Guesses) /* search the index of
    the password in the guess list. */
dist_index ← Index importance distribution
if Index >= 1 and Index <= K then
    GN ← Index
    return GN
Rank_R ← Zxcvbn_R(pw_A, Guesses) /* use the Zxcvbn_R
    module to calculate the Rank_R. */
Rank_G ← GuessSimRank(pw_A, Guesses, dist_index) /*
    use the GuessSimRank module to calculate the
    Rank_G. */
GN ← min(Rank_R, Rank_G) /* let the minimum value
    of Rank_R and Rank_G as GN. */
return GN
```

$Zxcvbn_R$ module utilize the basic Zxcvbn [60] and the guess list generated by POINTERGUESS. As shown in Fig. 17(a), the basic Zxcvbn is not designed for the target guessing scenarios and overlooks the strength of most passwords if given the old password. We incorporate the guess list based the user's old password to construct a new PSM that offers accurate evaluations in targeted guessing scenarios. More specifically, we employ sentencepiece [24] to extract popular password segments $Seg_{top}$ from the guess list. Then, we input $Seg_{top}$ into Zxcvbn to adjust the guess number evaluated by the basic Zxcvbn. We denote $Zxcvbn_R$ as

$$Zxcvbn_R\left(pw_A\right) = Zxcvbn\left(pw_A, \text{POINTERGUESS}\left(pw_A\right)\right), \tag{19}$$

where $\text{POINTERGUESS}\left(pw_A\right)$ denotes the guess list generated by POINTERGUESS using $pw_A$. Alg. 2 shows the detailed design of $Zxcvbn_R$ module.

**Algorithm 2:** *$Zxcvbn_R$* module

**Data:** Target password, Guess list, Segment count
**Result:** Guess Number *GN*

$L \leftarrow$ Segment count
$pw_A \leftarrow$ Target password
$BM \leftarrow$ BPE Method /* trained by sentencepiece [24], use it to split password. */
$Guesses \leftarrow$ Guess list
Define *split* split password function
Define *sort* sort segments by frequency function
*zxcvbn* // Zxcvbn function
$Seg \leftarrow split\,(BM, Guesses)$ /* split all guesses and statistic all segments. */
$Seg \leftarrow sort\,(Seg)$ /* sort by descending order of frequency. */
$GN \leftarrow zxcvbn(pw_A, Seg[0:L])$
**return** *GN*

Then, the *GuessSimRank* module uses a password similarity method *SimAlg* (e.g., edit distance) to calculate the similarity $sim_i$ of the $i$-th guess, $guess_i$, and the target password $pw_A$. That is $sim_i = SimAlg\,(pw_A, guess_i)$ Then we denote the guess number $Rank_G$ as

$$GuessSimRank\,(pw_A) = \sum_{i=1}^{K} W_{guess}\,(i) * SimGN\,(pw_A, sim_i, i), \tag{20}$$

where $K$ denotes the number of guesses, and $W_{guess}$ denotes the normalized *importance* distribution of different guesses in the guess list, weighting the impact of each $guess_i$ on $Rank_G$.

Note that the $SimGN\,(pw_A, sim_i, i)$ denotes that we use $sim_i$ and the guess number $i$ of $guess_i$ to calculate the guess number for $pw_A$. More specifically, we can express $SimGN\,(pw_A, sim_i, i)$ as

$$SimGN\,(pw_A, sim_i, i) = \exp\left((gn_A - i) * \sigma\,(sim_i)\right) + i, \tag{21}$$

where $\sigma(\cdot)$ is a sigmoid function, and $gn_A$ denote the guess number of $pw_A$ directly evaluated by Zxcvbn [60]. Alg. 3 shows the detailed design of *GuessSimRank* module. The results of the two modules are then compared, and the minimum value is selected as the final guess number *GN*.

## 7.2 Results and analysis

Fig. 17 evaluates the effectiveness of PR-PSM by investigating the password strength distributions of all target passwords (in the 89,951 unique test password pairs) with these 29,252 cracked target passwords in 12 scenarios. Fig. 17(a) shows that Zxcvbn [60] estimates the guess number for the majority of overall test passwords to exceed $10^5$, with over 15% of target passwords even deemed unguessable (i.e., $\geq 10^{10}$).

Zxcvbn accurately evaluates only a minority of the cracked passwords (i.e., guess number$\leq 10^3$) and overestimates the majority of the cracked passwords to exceed $10^5$, with some even exceeding $10^{10}$. In contrast, Fig. 17(b) shows that PR-PSM accurately evaluates all the cracked passwords to be within $10^3$ guesses, and the majority of overall test target passwords to be less than $10^{10}$. This indicates that Zxcvbn overestimates the strength of most passwords in targeted guessing scenarios, and PR-PSM well fixes its defect.

**Algorithm 3:** *GuessSimRank* module

**Data:** Target password, Guess list, Index importance distribution
**Result:** Guess Number *GN*

$pw_A \leftarrow$ Target password
*MLED* /* the function of calculating the Minimum Levenshtein Edit Distance */
$Guesses \leftarrow$ Guess list
*zxcvbn* // Zxcvbn function
$M \leftarrow zxcvbn\,(pw_A)$
$K \leftarrow \|Guesses\|$
$dist_{index} \leftarrow$ Index importance distribution
$rank \leftarrow 0$
**for** $i \leftarrow 1$ *to* $K$ **do**
  $score \leftarrow MLED\,(pw_A, Guesses[i])$ /* calculate the similarity score of target password and a guess */
  $importance \leftarrow dist_{index}[i]$
  $rank \leftarrow rank + importance *$ $SimGuessNum\,(pw_A, score, i)$ /* *SimGuessNum* is corresponding to Eq. 21 */
**return** *GN*

Table 7: Examples of password strengths by Zxcvbn [60] and PR-PSM.

| Index | Old password | Target password | $GN_Z$ † | $GN_R$ | *Hit* |
|---|---|---|---|---|---|
| 1 | yanlin19880911 | yanlin5201314 | 10.02 | 2.45 | Yes |
| 2 | yyt395746 | yyt3957460 | 10.00 | 1.08 | Yes |
| 3 | w564011 | 5640117aiolia | 12.11 | 8.29 | No |
| 4 | liu231377 | liujian231377 | 12.53 | 2.99 | No |
| 5 | gyhhx970621 | gylyx060504 | 9.16 | 5.62 | No |

† $GN_Z$ and $GN_R$ denote the guess number ($log_{10}$) evaluated by Zxcvbn [60] and PR-PSM, respectively. *Hit* denotes if the target password is cracked or not.

Table 7 presents five password pairs from the *Overall* list, demonstrating the difference in strength evaluations between Zxcvbn [60] and PR-PSM. As shown in Table 7, Zxcvbn seriously overestimates the strength of both cracked and uncracked passwords, while PR-PSM effectively uses the old password to evaluate password strength more accurately and provides a more reasonable evaluation (i.e., guess number). See more results and analysis in Appendix B of the full paper. Our experimental results demonstrate that designing a PSM that integrates the targeted guessing model can more accurately evaluate the strength of given passwords.

# 8 Conclusion

This paper provides a new technical route to dynamically generate a user's new password through her old password. For the first time, we propose a password reuse guessing model coupled with the pointer mechanism, namely POINTERGUESS. By introducing a hierarchical definition of password reuse, POINTERGUESS can characterize users' password reuse behaviors more accurately. Extensive experiments demonstrate the effectiveness of POINTERGUESS and its applicability to targeted PSMs. Furthermore, we investigate a realistic attack scenario where attackers leverage victims' multiple old passwords to compromise their current passwords, and propose MS-POINTERGUESS. We hope that our new models and accurate characterization of users' password reuse behaviors will help the academic community and web administrators have a better understanding of 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. 62222208 and 62172240, and Natural Science Foundation of Tianjin, China under Grant No. 21JCZDJC00190. See the full version of this paper at https://bit.ly/3wGx9Ke.

## References

[1] *Cracking passwords from the Mall.cz dump*, Jan. 2018, https://www.michalspacek.com/cracking-passwords-from-the-mall.cz-dump.

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

[3] *COMB data breach: what it means, and how to protect yourself*, Feb. 2021, https://blog.1password.com/what-comb-means-for-you-and-your-business/.

[4] *Recently added breaches.*, Dec. 2022, https://haveibeenpwned.com/.

[5] S. A. Al-Roomi and F. Li, "A large-scale measurement of website login policies," in *Proc. USENIX SEC 2023*, pp. 2061–2078.

[6] A. J. Aviv, D. Budzitowski, and R. Kuber, "Is bigger better? comparing user-generated passwords on 3x3 vs. 4x4 grid sizes for android's pattern unlock," in *Proc. ACSAC 2015*, pp. 301–310.

[7] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. ICLR 2015*.

[8] M. Bellare and G. Neven, "Multi-signatures in the plain public-key model and a general forking lemma," in *Proc. ACM CCS 2006*.

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

[10] 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.

[11] C. Castelluccia, A. Chaabane, M. Dürmuth, and D. Perito, *When privacy meets security: Leveraging personal information for password cracking*, Apr. 2013, https://arxiv.org/pdf/1304.6584.pdf.

[12] N. C. S. Centre, *Password managers: using browsers and apps to safely store your passwords*, Dec. 2021, https://www.ncsc.gov.uk/collection/top-tips-for-staying-secure-online/password-managers.

[13] C. Cimpanu, *Google launches Password Checkup feature, will add it to Chrome later this year.*, Oct. 2019, https://www.zdnet.com/article/google-launches-password-checkup-feature-will-add-it-to-chrome-later-this-year/.

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

[15] L. R. Dice, "Measures of the amount of ecologic association between species," *Ecology*, pp. 297–302, 1945.

[16] B. Fung, *Hackers post email addresses linked to 200 million Twitter accounts, security researchers say*, Jan. 2023, https://www.cnn.com/2023/01/05/tech/twitter-data-email-addresses/index.html.

[17] S. Gatlan, *Hacker leaks full database of 77 million Nitro PDF user records*, Jan. 2021, https://www.bleepingcomputer.com/news/security/hacker-leaks-full-database-of-77-million-nitro-pdf-user-records/.

[18] H. P. Google, *Online Security Survey*, Feb. 2019, https://services.google.com/fh/files/blogs/google_security_infographic.pdf.

[19] 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., Mar. 2020, https://pages.nist.gov/800-63-3/sp800-63b.html.

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

[21] X. He, H. Cheng, J. Xie, P. Wang, and K. Liang, "Passtrans: An improved password reuse model based on transformer," in *Proc. IEEE ICASSP 2022*, pp. 3044–3048.

[22] S. Katsumata, T. Matsuda, W. Nakamura, K. Ohara, and K. Takahashi, "Revisiting fuzzy signatures: Towards a more risk-free cryptographic authentication system based on biometrics," in *Proc. ACM CCS 2021*.

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

[24] T. Kudo, *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*, 2018, https://github.com/google/sentencepiece.

[25] LassPass, *The 2021 Password Security Report*, 2021, https://www.lastpass.com/resources/ebook/psychology-of-passwords-2021.

[26] W. LEVANDOWSKY, MICHAEL and DAVID, "Distance between sets," *Nature*, pp. 34–35, 1971.

[27] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," *Soviet physics. Doklady*, pp. 707–710, 1965.

[28] Y. Li, Y. Li, X. Chen, R. Shi, and J. Han, "Pg-pass:targeted online password guessing model based on pointer generator network," in *Proc. IEEE CSCWD 2022*, pp. 507–512.

[29] Z. Li, W. He, D. Akhawe, and D. Song, "The emperor's new password manager: Security analysis of web-based password managers," in *Proc. USENIX SEC 2014*, pp. 465–479.

[30] Lily Hay Newman, *The Password Isn't Dead Yet. You Need a Hardware Key*, Dec. 2022, https://www.wired.com/story/hardware-security-key-passwords-passkeys/.

[31] N. Lord, *Uncovering Password Habits: Are Users' Password Security Habits Improving? (Infographic).*, Sep. 2020, https://digitalguardian.com/blog/uncovering-password-habits-are-users-password-security-habits-improving-infographic/.

[32] 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.

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

[34] C. Morris, *Massive data leak exposes 700 million LinkedIn users' information*, Jun. 2021, https://fortune.com/2021/06/30/linkedin-data-theft-700-million-users-personal-information-cybersecurity/.

[35] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *J. Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970.

[36] E. Nesbo, *8 Reasons Password Managers Are Not as Safe as You Think*, May. 2022, https://www.makeuseof.com/are-password-managers-safe-or-not/.

[37] L. H. Newman, *Why the Password Isn't Dead Quite Yet*, Jul. 2021, https://www.wired.com/story/passwords-not-dead-yet-authentication/amp.

[38] 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.

[39] 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 2021*, pp. 1831–1848.

[40] 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.

[41] 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*.

[42] S. Pearman, S. A. Zhang, L. Bauer, N. Christin, and L. F. Cranor, "Why people (don't) use password managers effectively," in *Proc. USENIX SEC 2019*, pp. 319–338.

[43] S. Perkins, *The 5 top reasons you should use a password manager*, Feb. 2023, https://www.androidpolice.com/top-reasons-download-use-password-manager/.

[44] V. Petkauskas, *Mother of all breaches reveals 26 billion records: what we know so far*, Jan. 2024, https://cybernews.com/security/billions-passwords-credentials-leaked-mother-of-all-breaches/.

[45] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," in *Proc. ACL 2017*, pp. 1073–1083.

[46] M. Shirvanian and S. Agrawal, "2D-2FA: A new dimension in two-factor authentication," in *Proc. ACSAC 2021*, pp. 482–496.

[47] P. Shrestha, A. T. Mahdad, and N. Saxena, "Sound-based two-factor authentication: Vulnerabilities and redesign," *ACM Trans. Priv. Secur.*, vol. 27, no. 1, pp. 1–27, 2023.

[48] 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, pp. 1929–1958, 2014.

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

[50] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. NeurIPS 2015*, pp. 2692–2700.

[51] C. Wang, J. Zhang, M. Xu, H. Zhang, and W. Han, "# Segments: A Dominant Factor of Password Security to Resist against Data-Driven Guessing," *Comput. Secur.*, vol. 121, p. 102848, 2022.

[52] 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.

[53] 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, http://bit.ly/2ahJ8CO.

[54] D. Wang, X. Shan, Q. Dong, Y. Shen, and C. Jia, "No single silver bullet: Measuring the accuracy of password strength meters," in *Proc. USENIX SEC 2023*, pp. 947–964.

[55] 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. 1537–1554.

[56] 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.

[57] D. Wang, Y. Zou, Y.-A. Xiao, S. Ma, and X. Chen, "PASS2EDIT: A multi-step generative model for guessing edited passwords," in *Proc. USENIX SEC 2023*, pp. 9803–1000.

[58] D. Wang, Y. Zou, Z. Zhang, and K. Xiu, "Password guessing using random forest," in *Proc. USENIX SEC 2023*, pp. 965–982.

[59] 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.

[60] D. Wheeler, "zxcvbn: Low-budget password strength estimation," in *Proc. USENIX SEC 2016*, pp. 157–173.

[61] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proc. ACL 2020*, pp. 38–45.

[62] XposedOrNot, *Exposed Data Breaches*, 2024, https://xposedornot.com/xposed.

[63] A. Yee, *6 common reasons people don't use a password manager—and why they're wrong*, Jun. 2022, https://www.pcworld.com/article/709052/6-common-reasons-people-dont-use-a-password-manager-and-why-theyre-wrong.html/.

# A Supplementary details of other models

We now introduce several state-of-the-art models and alternative models that serve as benchmarks for comparison.

**TarGuess-II**. TarGuess-II was proposed by Wang et al. at CCS'16 [56]. This model is based on PCFG [59] for training a probabilistic structure model. Additionally, it incorporates an n-gram Markov model [33] to generate two n-gram files, one in the original order and the other in reverse. Moreover, TarGuess-II models users' transformation behaviors at segment- and character-level. When generating guesses, TarGuess-II mixes guesses with a popular password dictionary. We denote it as $Top_{cn}$ (for Chinese) and $Top_{en}$ (for English). In this paper, we use CSDN, Dodonew, and 126 to compose $Top_{cn}$ and use 000Webhost, LinkedIn, and Yahoo to compose $Top_{en}$. Then, we multiply the ranking by the frequency of passwords in the three datasets and chose Top-$10^4$ as the dictionary. Note that for all parameters of this model, we keep the default settings provided by the authors.

**Pass2Path**. In their paper presented at IEEE S&P'19, Pal et al. [38] introduced Pass2Path, a targeted guessing model based on seq2seq [49]. Pass2Path is designed to complete the "password-to-path" task (i.e., take character sequence as input and the edit-operation sequence as output) and generate new passwords based on generated operation sequences. The model uses a three-layer RNNs with a hidden dimension of 128 for both the encoder and decoder. They set the learning rate to 0.0003 and the dropout rate to 0.4. We follow the same configuration for the validation set as with [38]. As Pass2Path is open-source, we used the same model structure and the settings mentioned above as recommended by Pal et al. [38].

**PlainSeq**. At IEEE S&P'19, Pal et al. [38] also introduced Pass2Pass, a targeted guessing model tailored for the "password-to-password" task. Inspired by their work, we design a similar model, which we refer to as PlainSeq. In this paper, we use PlainSeq to conduct an ablation study to demonstrate the effectiveness of the pointer mechanism. We only keep the basic sequence-to-sequence model. It is worth noting that we did not incorporate the `<key-sequence>` proposed by Pal et al. [38] into PlainSeq. Instead, we used the original password string (i.e., character sequence) as model input to make a fair comparison with our POINTERGUESS.

**Pass2Edit**. Wang et al. [57] proposed a new algorithm called Pass2Edit. They redefined the password generation task as a "multi-step decision classification" task. Pass2Edit has a 3-layer GRUs and two fully connected layers. At each timestep, Pass2Edit takes the modified password and the original password (at the character level) as input and predicts one atomic edit operation which will be applied to the modified password. As the source code provided to us by Pass2Edit's authors, we keep the model's default settings and only adjust it on training/test datasets. Note that in [57] Want et al. proposed two models, Pass2Edit-nomix and Pass2Edit-mix. Pass2Edit-nomix is the original model that does not mix an extra popular password dictionary, while Pass2Edit-mix heuristically mixes its guessing list with popular passwords to output the final guessing list. In this paper, we name the model without mixing popular passwords (i.e., Pass2Edit-nomix) as Pass2Edit.

**Untargeted dictionary attack (Top-PW)**. We build two popular password dictionaries based on Chinese (i.e., CSDN, 126, Dodonew) and English (i.e., 000Webhost, LinkedIn, Yahoo) training sets. We first sort these passwords in descending order of frequency. Then we select Top-$10^3$ as the popular dictionary (as our maximum guess number is $10^3$), then we use them to build an untargeted dictionary attack.

**Mixing models with an extra popular password dictionary**. Note that the models mentioned above refer to the original models that are not mixed with popular passwords, except for TarGuess-II [56]. We investigate how mixing popular passwords with the original model outputs affects each model's performance. We follow the same strategy of mixing popular passwords as proposed in [57], and append "-mix" to the original model's name to denote the model using the mixing strategy. Furthermore, we strive to keep the random seeds used for these neural network-based models consistent with those used in the original paper (e.g., Pass2Edit [57]) to ensure more meaningful and reliable comparisons.

Note that the PG-Pass model proposed by Li et al. [28] is unsuitable for comparison with our POINTERGUESS, because these two models are initially designed for different attacking scenarios: PG-Pass focuses on PII-based targeted guessing scenarios, while our POINTERGUESS focuses on password reuse-based targeted guessing scenarios.

# B Potential applications

**Password protection**. Here, we discuss two potential approaches for integrating POINTERGUESS into compromised credential checking (C3) services, such as MIGP, drawing inspiration from the application of Pass2Path [38] in MIGP [39]. First, we directly apply our POINTERGUESS to dynamically generate guesses. We denote this approach as "PTG". Second, as the "wEdit" proposed in [39], we explore using POINTERGUESS to generate a ranked list of tweaks, which can be applied to generate guesses. We denote this approach as "PTG-wEdit". Below we briefly introduce these two approaches.

First, "PTG" involves the real-time execution of POINTERGUESS within MIGP. Here, the pre-trained POINTERGUESS model is loaded onto the server's CPU/GPU. When a client submits the target password, the loaded model is immediately utilized to generate a set of password variants. This approach capitalizes on the ability to dynamically generate password variants using a pre-trained model. However, while offering flexibility and on-the-fly variant generation, this method comes with drawbacks such as increased resource consumption and potential speed issues during protocol execution.

Second, we employ a pre-trained POINTERGUESS to produce a series of password guesses for each user. These guesses are then transformed into the "transformation path" format proposed by Pal et al. [38]. By using a given dataset, POINTERGUESS generates transformation paths for all password pairs. We tally the occurrence of each transformation path and rank them in descending order based on frequency. While generating $n$ variants of the target password, we apply the first $n$ valid transformation paths from the sorted list. This approach offers the advantage of requiring fewer computational resources and providing faster variant generation. However, it is important to note that it relies solely on a statistically derived list of transformation paths for generating variants.

Overall, our proposed approaches aim to enhance password protection within the MIGP framework by leveraging the capabilities of POINTERGUESS. The first approach offers dynamic variant generation using a pre-trained model but may demand higher computational resources and raise speed concerns. In contrast, the second approach enables rapid and resource-efficient variant generation based on a statistically derived list of transformation paths. Further exploration and evaluation are necessary to gauge the feasibility and suitability of these approaches, considering the specific requirements and constraints of the MIGP system. Balancing execution speed and accuracy emerges as a critical issue to address in future research endeavors.