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.
**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 POINT-ERGUESS. 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-POINTER GUESS, which can employ the victim’s multiple leaked passwords.

By employing 13 large-scale real-world password datasets, we demonstrate that POINT-ERGUESS is effective: (1) When the victim’s password at site A (namely pwA) is known, within 100 guesses, the average success rate of POINT-ERGUESS in guessing her password at site B (namely pwB, pwA ≠ pwB) 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., pwA can equal pwB), within 100 guesses, the average success rate of POINT-ERGUESS 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-POINTER GUESS further improves the cracking success rate by 31.21% compared to POINT-ERGUESS.

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 vulnerability.

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 $\langle \text{BOS} \rangle, (\text{Del}, 0, 'I'), (\text{Del}, 1, 'I'), (\text{Del}, 2, 'o'), (\text{Del}, 3, 'v'), (\text{Del}, 4, 'e'), (\text{Ins}, 10, '6'), (\text{Ins}, 10, '7'), (\text{Ins}, 10, '8'), (\text{Ins}, 10, '9'), <\text{EOS}>\rangle$, where $\langle \text{Del}, 0, 'I' \rangle$ denotes deleting the character ‘I’ in the first position of $pw_A$, and $\langle \text{BOS} \rangle$ 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 $\leq 4$ for training to avoid the negative impacts of future/distant password pairs like yjqqq916198 $\rightarrow$ 916198yj). This makes it difficult for them to generate long (yet realistic) edit sequences (e.g., edit distance $\geq 5$), 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, $P_{\text{copy}}$ and $P_{\text{vocab}}$, respectively. $<\text{BOS}>/<\text{EOS}>$ represents the start/end of the generation.

---

**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 cannot 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 filter out “dissimilar” password pairs during training, and thus they cannot 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 employs 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, POINTER GUESS. By leveraging the pointer mechanism, POINTER GUESS 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-POINTER GUESS, to assess the threat of attackers using multiple old passwords to compromise the target password. By hierarchically re-defining “password reuse” on two levels, we demonstrate the effectiveness of POINTER GUESS and provide a new angle to understand the performance of existing models.

- **Extensive evaluation.** We demonstrate the effectiveness of POINTER GUESS on 12 practical attack scenarios by employing 11 large-scale password datasets. More specifically, within 100 guesses, POINTER GUESS 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-POINTER GUESS over our POINTER GUESS in two practical attack scenarios. More specifically, within 100 guesses, the MS-POINTER GUESS achieves a success rate 31.21% (on average) higher than POINTER GUESS.

- **A password reuse-based password strength meter.** We introduce a password reuse-based password strength meter, called PR-PSM, by integrating Zxcvbn [60] with POINTER GUESS 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 POINTER GUESS can capture complex password reuse behaviors (e.g., $1991322322 \rightarrow 1.99132E+12$). These findings enhance our understanding of password reuse and showcase the effectiveness of POINTER GUESS. 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

$$P(pw_B|pw_A) = P(e'_1, e'_2, ..., e'_M|c_1, c_1, ..., c_N) = \prod_{i=1}^{M} P(e'_i|pw_A, c_{c_i}).$$

(1)

where $c_{c_i} = (e'_1, ..., e'_{i-1})$ denotes the subsequence of $pw_B$, and how to model $P(e'_i|pw_A, c_{c_i})$ 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

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

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, POINTER GUESS 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 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 ø). 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 POINTER GUESS, which is based on [45, 50] and consists of a basic seq2seq model [49] and a pointer module. $P_\text{copy}$ (i.e., “Final Distribution”) denotes the conditional password probability of POINTER GUESS generating the next character. $P_\text{gen}$ and $P_\text{vocab}$ are the conditional probabilities of copying characters from the old password, and of generating new characters from the vocabulary, respectively. POINTER GUESS employs a soft switch $p_\text{gen}$ 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., KevelMobile → password123). Furthermore, it extends to reusing some specific popular password segments, like KevelMobile → 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., $P_\text{vocab}(M) = 0.005$). 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 POINTER GUESS incorporates $P_\text{vocab}$ (the conditional probability of generating characters from vocabulary, i.e., 95 printable characters) and $P_\text{copy}$ (the conditional probability of copying from the old password), to model the conditional password probability ($P_\text{gen}$). As shown in Fig. 1, after incorporating $P_\text{copy}$, the probability of generating ‘M’ significantly increases (i.e., $P_\text{gen}(M) = 0.097 > P_\text{vocab}(M) = 0.005$). In each subsequent step, POINTER GUESS dynamically adjusts the probabilities of each character through $P_\text{copy}$ and $P_\text{vocab}$, gradually approaching the correct target password (i.e., MacOP6789). Below, we formally describe how POINTER GUESS models the conditional password probability.

As shown in Fig. 3, at each timestep $t$, POINTER GUESS 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 POINTER GUESS outputs encoder hidden states $h_t$ and decoder hidden states $s_t$. Subsequently, POINTER GUESS computes the conditional probability of generating characters from the vocabulary, i.e., $P_\text{vocab}$, as:

$$P_\text{vocab} = \text{softmax} \left( W' \left( W \ast [s_t, c_t] + b_{\text{out}} \right) + b_{\text{out}}' \right),$$

where $[s_t, c_t]$ denotes concatenating $s_t$ and $c_t$, $W, W'$, $b_{\text{out}}$, $b_{\text{out}}'$ 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, POINTER GUESS 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., $P_\text{copy}$, which can be expressed as:

$$P_\text{copy}(c) = FFN \left( \sum_{j : c_j = c} a_j \right),$$

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

To facilitate flexible decision-making on whether to copy characters from the old password or generate new characters from the vocabulary, POINTER GUESS utilizes a soft switch $p_\text{gen}$, which is:

$$p_\text{gen} = \sigma(W_x \ast c_t + W_s \ast s_t + W_y \ast y_t + b_g),$$

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

Finally, POINTER GUESS integrates $P_\text{vocab}$ and $P_\text{copy}$ to generate $P_\text{gen}$, representing the conditional password probability $P_\text{gen}$, which is expressed as:

$$P_\text{gen}(c) = p_\text{gen} \ast P_\text{copy}(c) + (1 - p_\text{gen}) \ast P_\text{vocab}(c).$$

This allows POINTER GUESS 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, POINTER GUESS consists of three phases: Preprocess, training, and generation. We now present the workflow of POINTER GUESS that tackles model training and guess generation, and detail the preprocess phase in Sec. 4.

#### Model architecture

As shown in Fig. 3, our POINTER GUESS 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
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 \(L\), which measures the log probability of \(\hat{Y}_{BS}\) being aligned with the ground truth passwords \(Y_{BS}\). The goal of \(L\) is to find the optimized parameters \(\theta^*\), which are

\[
\theta^* = \arg \min_{\theta} \left( L_\theta \left( Y_{BS}, \hat{Y}_{BS} \right) \right),
\]

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 \(1 = e-12\) as our smoothed value. We denote the vocabulary as \(\Sigma\), consisting of 95 printable ASCII characters and four special identifiers (i.e., \(<EOS>\), \(<POS>\), \(<PAD>\), 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
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 only contain 95 printable ASCII characters. We call this empty/invalid email. We also keep passwords that are malicious parties for misconduct, while our use is beneficial for the community to understand password strength and for more informed decisions.

### Dataset cleaning
First, we remove data items with an empty/invalid email. We also keep passwords that are less than 30 characters long 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~#8 and #5~#8 for Chinese and English users, respectively. All selected test sets, except for the scenario #8, are in plaintext. At 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 the attack scenarios.

<table>
<thead>
<tr>
<th>#</th>
<th>Attack scenario</th>
<th>Language</th>
<th>Training set setup</th>
<th>Size (pairs)</th>
<th>Testing set setup</th>
<th>Size (pairs)</th>
<th>Clean strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>126 → CSDN</td>
<td>Chinese</td>
<td>Tianya, Dodonew</td>
<td>188,928</td>
<td>126 → CSDN</td>
<td>83,206</td>
<td>Len↑5</td>
</tr>
<tr>
<td>#2</td>
<td>CSDN → 126</td>
<td>Chinese</td>
<td>Tianya → CSDN</td>
<td>211,385</td>
<td>CSDN → 126</td>
<td>86,104</td>
<td>Basic</td>
</tr>
<tr>
<td>#3</td>
<td>Tianya → CSDN</td>
<td>Chinese</td>
<td>CSDN → Tianya</td>
<td>434,255</td>
<td>Tianya → CSDN</td>
<td>826,599</td>
<td>Len↑5</td>
</tr>
<tr>
<td>#4</td>
<td>CSDN → Dodonew</td>
<td>Chinese</td>
<td>CSDN → Dodonew</td>
<td>86,104</td>
<td>CSDN → Dodonew</td>
<td>211,385</td>
<td>Basic</td>
</tr>
<tr>
<td>#5</td>
<td>000Webhost → LinkedIn</td>
<td>English</td>
<td>000Webhost → LinkedIn</td>
<td>265,381</td>
<td>000Webhost → LinkedIn</td>
<td>213,597</td>
<td>Len↑5</td>
</tr>
<tr>
<td>#6</td>
<td>Yahoo → 000Webhost</td>
<td>English</td>
<td>Yahoo → 000Webhost</td>
<td>40,646</td>
<td>Yahoo → 000Webhost</td>
<td>37,479</td>
<td>Len↑5</td>
</tr>
<tr>
<td>#7</td>
<td>LinkedIn → 000Webhost</td>
<td>English</td>
<td>LinkedIn → 000Webhost</td>
<td>40,812</td>
<td>LinkedIn → 000Webhost</td>
<td>259,175</td>
<td>Len↑5</td>
</tr>
<tr>
<td>#8</td>
<td>000Webhost → RedMart</td>
<td>English</td>
<td>000Webhost → RedMart</td>
<td>213,697</td>
<td>000Webhost → RedMart</td>
<td>6,858</td>
<td>Len↑5</td>
</tr>
<tr>
<td>#9</td>
<td>80% Mixed_EN → 20% Mixed_CN</td>
<td>Chinese</td>
<td>80% of Mixed_EN</td>
<td>434,255</td>
<td>20% of Mixed_EN</td>
<td>84,714</td>
<td>Basic</td>
</tr>
<tr>
<td>#10</td>
<td>80% Mixed_CN → 20% Mixed_EN</td>
<td>Chinese</td>
<td>80% of Mixed_CN</td>
<td>434,255</td>
<td>20% of Mixed_CN</td>
<td>108,564</td>
<td>Basic</td>
</tr>
<tr>
<td>#11</td>
<td>80% 4iQ → 20% 4iQ</td>
<td>Mixed</td>
<td>80% of 4iQ dataset</td>
<td>116,837,808</td>
<td>20% 4iQ dataset</td>
<td>29,209,452</td>
<td>Basic</td>
</tr>
<tr>
<td>#12</td>
<td>80% COMB → 20% COMB</td>
<td>Mixed</td>
<td>80% of COMB</td>
<td>342,921,727</td>
<td>20% COMB dataset</td>
<td>85,730,432</td>
<td>Basic</td>
</tr>
<tr>
<td>#13A</td>
<td>Tianya, 126 → Taobao</td>
<td>Chinese</td>
<td>Tianya, 126 → Taobao</td>
<td>95,457</td>
<td>Tianya, 126 → Taobao</td>
<td>79,562</td>
<td>Basic</td>
</tr>
<tr>
<td>#13B</td>
<td>Tianya → Taobao</td>
<td>Chinese</td>
<td>Tianya → Taobao</td>
<td>126 → Taobao</td>
<td>79,562</td>
<td>Basic</td>
<td></td>
</tr>
<tr>
<td>#13C</td>
<td>126 → Taobao</td>
<td>Chinese</td>
<td>126 → Taobao</td>
<td>79,562</td>
<td>Basic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#14A</td>
<td>80% UnionA → 20% UnionA</td>
<td>Chinese</td>
<td>80% of UnionA dataset</td>
<td>27,833,899</td>
<td>20% UnionA dataset</td>
<td>10,785,542</td>
<td>Basic</td>
</tr>
<tr>
<td>#14B</td>
<td>80% UnionA → 20% UnionB</td>
<td>Chinese</td>
<td>80% of UnionA dataset</td>
<td>27,833,899</td>
<td>20% UnionB dataset</td>
<td>10,785,542</td>
<td>Basic</td>
</tr>
<tr>
<td>#14C</td>
<td>80% UnionA → 20% UnionB</td>
<td>Chinese</td>
<td>80% of UnionA dataset</td>
<td>27,833,899</td>
<td>20% UnionA dataset</td>
<td>10,785,542</td>
<td>Basic</td>
</tr>
</tbody>
</table>

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

‡ A → B (e.g., #1. 126 → CSDN) means that: A user’s password at service A can be used by an attacker to help attack user’s password at service B. Note that for #13A and #14A, we represent the attack scenarios as A1, A2 → 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 → 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 and LinkedIn. (2) The RedMart dataset is built by matching ClixSense, LiveAuctioneers and 4iQ using email. Note that we remove any data item with fewer than three passwords.

†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.
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 POINTER GUESS achieves the highest cracking success rate in 10 of 12 scenarios over its foremost counterparts TarGuess-II [56], Pass2Edit [57] and Pass2Path [38].

POINTER GUESS with MS-POINTER GUESS.

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 UnionA1 and UnionA2 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 tawling 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 POINTER GUESS 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., pwA = pwB) 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.
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 POINTER GUESS 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 POINTER GUESS 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 POINTER GUESS 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 POINTER GUESS 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 POINTER GUESS 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 POINTER GUESS 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 POINTER GUESS, and find that POINTER GUESS 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 POINTER GUESS in Chinese (#1∼#4) and English (#5∼#8) attack scenarios. As shown in Fig. 6, the experimental results demonstrate that POINTER GUESS outperforms other models in all Chinese scenarios. In English scenarios, our model still achieves higher or comparable performance. Notably, POINTER GUESS significantly outperforms other models when attacking security-savvy users (see scenarios #6 and #7 that attack 000webhost).

It is worth noting that our POINTER GUESS 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 POINTER GUESS 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 POINTER GUESS) 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 POINTER GUESS-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 POINTER GUESS-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 POINTER GUESS-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 POINTER GUESS-mix are 2.04% and 8.89% (on average) higher than Pass2Edit-mix and Pass2Path-mix, respectively.

**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...
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).

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
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 POINTER GUESS shows a clear advantage in cracking such password pairs. We can see that another 40% of password pairs still cannot be identified as reuse (see “Total” when x-axis=“Others”) by POINTER GUESS and other major models. This outlines the need for a more thorough understanding of users’ reuse behaviors.

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

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>
<thead>
<tr>
<th>Attack model</th>
<th>POINTER GUESS</th>
<th>Pass2Edit [57]</th>
<th>Pass2Path [38]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training time</td>
<td>15:14</td>
<td>09:43</td>
<td>14:10</td>
</tr>
<tr>
<td>Testing time</td>
<td>00:24</td>
<td>02:26</td>
<td>01:47</td>
</tr>
<tr>
<td>Speed (pw/s)</td>
<td>9,700~9,800</td>
<td>2,100~2,200</td>
<td>2,900~3,300</td>
</tr>
<tr>
<td>Model size (MB)</td>
<td>2.26</td>
<td>11</td>
<td>53.6</td>
</tr>
</tbody>
</table>

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, POINTER GUESS 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 POINTER GUESS 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 ≥0.5. To address this issue, we intro-
duce our MS-POINTER GUESS 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 POINTER GUESS 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 POINTER GUESS 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 POINTER GUESS demonstrates clear preferences in complex reuse behaviors. First, POINTER GUESS can characterize users’ vulnerable behaviors of reusing popular passwords (e.g., abc123 in the first example). Second, POINTER GUESS can generate semantic fragments based on the old password (e.g., email suffixes like @hotmail.com), potentially assisting in guessing reused passwords. Third, POINTER GUESS 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 POINTER GUESS: 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-POINTER GUESS. MS-POINTER GUESS incorporates a “Multi-Encoder” module into POINTER GUESS 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-POINTER GUESS to effectively characterize users’ password reuse behaviors by leveraging multiple contexts.

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

In general, the conditional guessing probability that an attacker exploits users’ multiple leaked passwords (e.g., pwA at site A and pwB at site B) to attack the victim’s target password at site C (namely pwC = (c′₁,…,c′₁₉)) can be expressed similarly to Eq. 1, which is

\[ P(pwC|pw_A,pw_B) = \prod_{i=1}^{M} P(c'_{i}|c'_{<i},pw_A,pw_B), \]

where \( c'_{<i} \) dotes the sub-sequence \( (c'_{0},...,c'_{i-1}) \) of \( pw_C \), and \( P(c'_{i}|c'_{<i},pw_A,pw_B) \) 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 POINTER GUESS 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-POINTER GUESS outputs two context vectors, \( c_t, c_{t'} \), from \( pw_A \)
and $pw_B$, respectively. Then, we can express $\lambda \in [0, 1]$ as

$$\lambda = \sigma \left( W_c \cdot c_t + W'_c \cdot c'_t + b_\lambda \right),$$  

(9)

where $\sigma$ 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., $P_{copy}^A$) and $pw_B$ (i.e., $P_{copy}^B$), respectively. This yields $P_{copy}$, the weighted conditional probability of copying characters from $pw_A$ and/or $pw_B$, can be expressed as:

$$P_{copy}(c) = \lambda \cdot P_{copy}^A(c) + (1 - \lambda) \cdot P_{copy}^B(c).$$  

(10)

Then, we can represent $P_{copy}^A$ and $P_{copy}^B$, similarly to Eq. 4, which are

$$P_{copy}^A(c) = FFN \left( \sum_{t \in c} \alpha'_t \right),$$  

(11)

and

$$P_{copy}^B(c) = FFN \left( \sum_{t \in c} \alpha''_t \right),$$  

(12)

where $\alpha'_t$ and $\alpha''_t$ 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 $\sum_{t \in c} \alpha'_t$ or $\sum_{t \in c} \alpha''_t$ will be zero.

Still, we use the pointer mechanism $p_g$ to weight $P_{copy}(c)$ 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

$$p_g = \sigma \left( W_c \cdot c_t + W'_c \cdot c'_t + W_s \cdot s_t + W_x \cdot x_t + b_g \right),$$  

(13)

where $W_c$, $W'_c$, $W_s$, $W_x$, $b_g$ are learnable parameters, and $\sigma(\cdot)$ is a sigmoid function.

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

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

(14)

Overall, the scalability facilitated by the “Multi-Encoder” module in MS-POINTER GUESS empowers our model to handle users’ multiple leaked passwords simultaneously. This capability enables MS-POINTER GUESS 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-POINTER GUESS dynamically determines whether to copy characters from the old passwords or generate new characters directly from the vocabulary.

### 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-POINTER GUESS. Fig. 15 shows that MS-POINTER GUESS invariably outperforms POINTER GUESS across both attack scenarios. For identical password pairs, MS-POINTER GUESS achieves cracking success rates that are, on average, 17.54% (scenarios #13) and 26.11% (scenarios #14) higher than POINTER GUESS within 100 guesses, respectively. Even when excluding identical password pairs, MS-POINTER GUESS 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 POINTER GUESS, respectively. See Table 6 for more details and specific results.

This highlights the effectiveness of MS-POINTER GUESS and the substantial impacts of utilizing different training sets to attack the same test set on POINTER GUESS’ 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, MS-POINTER GUESS achieves cracking rates that are, on average, 17.54% (scenarios #13) and 26.11% (scenarios #14) higher than POINTER GUESS, respectively. See Table 6 for more details and specific results.

#### 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.
To address this limitation, we introduce PR-PSM, a password reuse-based PSM that integrates our POINTER GUESS 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 ZxcvbnR 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 pwB into the targeted password guessing model (e.g., POINTER GUESS) and generate Top-\(K\) guesses. Then, if pwA is in the guess list, we use its index in the guess list as the guess number, that is

\[
GN = \text{Index}(pw_A).
\]

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, ZxcvbnR, integrates Zxcvbn [60] and evaluates the strength \(Rank_R\) of \(pw_A\) using the generated guess list, which can be expressed as

\[
Rank_R = \text{ZxcvbnR}(pw_A).
\]

The second module, GuessSimRank, integrates POINTER GUESS and Zxcvbn to evaluate the strength \(Rank_G\) of \(pw_A\), that is

\[
Rank_G = \text{GuessSimRank}(pw_A).
\]

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

\[
GN = \min(Rank_R, Rank_G).
\]

Alg. 1 shows the detailed design of PR-PSM. Then we describe how we use ZxcvbnR and GuessSimRank two modules to reevaluate the strength of the target password \(pw_A\).

\begin{algorithm}
\caption{PR-PSM evaluation}
\begin{algorithmic}
\Data Target password, Targeted password guessing model, \(K\), Index importance distribution
\Result Guess Number \(GN\)
\Function{alg1}{\Input \(pw_A\) \rightarrow Target password \Model \leftarrow Targeted password guessing model /* input password and guess number, output guesses */ \Guesses \leftarrow Model(\(pw_A\), \(K\)) /* get the guess list, which has \(K\) guesses. */ \Index \leftarrow \text{Search}(pw_A, \text{Guesses}) /* search the index of the password in the guess list. */ \distindex \leftarrow \text{Index importance distribution} \If {\Index >= 1 and \Index <= \(K\)} \State \(GN \leftarrow \Index\) \State \text{return} \(GN\)
\EndIf
\RankR \leftarrow \text{ZxcvbnR}(pw_A, \text{Guesses}) /* use the ZxcvbnR module to calculate the \RankR. */ \RankG \leftarrow \text{GuessSimRank}(pw_A, \text{Guesses}, \distindex) /* use the GuessSimRank module to calculate the \RankG. */ \State \(GN \leftarrow \min(\RankR, \RankG)\) /* let the minimum value of \RankR and \RankG as \(GN\). */ \State \text{return} \(GN\)
\EndFunction
\end{algorithmic}
\end{algorithm}

ZxcvbnR module utilize the basic Zxcvbn [60] and the guess list generated by POINTER GUESS. 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 on 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 ZxcvbnR as

\[
\text{ZxcvbnR}(pw_A) = \text{Zxcvbn}(pw_A, \text{POINTERGUESS}(pw_A)),
\]

where \(\text{POINTERGUESS}(pw_A)\) denotes the guess list generated by POINTER GUESS using \(pw_A\). Alg. 2 shows the detailed design of ZxcvbnR module.
Algorithm 2: ZxcvbnR module

**Data:** Target password, Guess list, Segment count

**Result:** Guess Number GN

1. $L \leftarrow$ Segment count
2. $pw_A \leftarrow$ Target password
3. $BM \leftarrow$ BPE Method /* trained by sentencepiece [24], use it to split password. */
4. $Guesses \leftarrow$ Guess list
5. Define $split$ split password function
6. Define $sort$ sort segments by frequency function
7. $zxcvbn // Zxcvbn function$
8. $Seg \leftarrow$ split ($BM, \text{Guesses}$) /* split all guesses and statistic all segments. */
9. $Seg \leftarrow$ sort ($Seg$) /* sort by descending order of frequency. */
10. $GN \leftarrow$ zxcvbn ($pw_A, Seg[0:L]$)
11. 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

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

(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 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

$$
SimGN (pw_A, sim_i, i) = \exp ((gn_A - i) \times \sigma (sim_i)) + i,
$$

(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 $10^5$ 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.

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.

<table>
<thead>
<tr>
<th>Index</th>
<th>Old password</th>
<th>Target password</th>
<th>GNz</th>
<th>GNg</th>
<th>Hit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>yanlin57880911</td>
<td>yanlin5201314</td>
<td>10.02</td>
<td>2.46</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>yyi395746</td>
<td>yyi395746</td>
<td>10.00</td>
<td>1.08</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>$w66012$</td>
<td>$w640171011$</td>
<td>12.11</td>
<td>8.29</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>lu2631977</td>
<td>lu2631977</td>
<td>12.53</td>
<td>2.99</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>qjhx970621</td>
<td>qjhx970621</td>
<td>9.16</td>
<td>5.62</td>
<td>No</td>
</tr>
</tbody>
</table>

$GNz$ and $GNg$ denote the guess number ($log_{10}$) evaluated by Zxcvbn [60] and PR-PSM, respectively. Hit denotes if the target password is cracked or not.
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 62127240, and Natural Science Foundation of Tianjin, China under Grant No. 21JCZDJC00190. See the full version of this paper at https://bit.ly/3wGx9Ke.

References

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 choose 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 POINTER GUESS.

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-103 as the popular dictionary (as our maximum guess number is 103), 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 POINTER GUESS, because these two models are initially designed for different attacking scenarios: PG-Pass focuses on PII-based targeted guessing scenarios, while our POINTER GUESS focuses on password reuse-based targeted guessing scenarios.

B Potential applications

Password protection. Here, we discuss two potential approaches for integrating POINTER GUESS 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 POINTER GUESS to dynamically generate guesses. We denote this approach as “PTG”. Second, as the “wEdit” proposed in [39], we explore using POINT-ERGUESS 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 POINTER GUESS within MIGP. Here, the pre-trained POINTER GUESS 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 POINTER GUESS 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, POINTER GUESS 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 POINTER GUESS. 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.