Improving Real-world Password Guessing Attacks via Bi-directional Transformers

Ming Xu and Jitao Yu, *Fudan University;* Xinyi Zhang, *Facebook;* Chuanwang Wang, Shenghao Zhang, Haoqi Wu, and Weili Han, *Fudan University*

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-ming

This paper is included in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

Improving Real-world Password Guessing Attacks via Bi-directional Transformers

Ming Xu[1], Jitao Yu[1], Xinyi Zhang[2], Chuanwang Wang[1], Shenghao Zhang[1], Haoqi Wu[1], and Weili Han[1]

[1]Laboratory for Data Analytics and Security, Fudan University
[2]Applied AI Research, Facebook

Abstract

Password guessing attacks, prevalent issues in the real world, can be conceptualized as efforts to approximate the probability distribution of text tokens. Techniques in the natural language processing (NLP) field naturally lend themselves to password guessing. Among them, bi-directional transformers stand out with their ability to utilize bi-directional contexts to capture the nuances in texts.

To further improve password guessing attacks, we propose a bi-directional-transformer-based guessing framework, referred to as PassBERT, which applies the *pre-training / fine-tuning* paradigm to password guessing attacks. We first prepare a pre-trained password model, which contains the knowledge of the general password distribution. Then, we design three attack-specific fine-tuning approaches to tailor the pre-trained password model to the following real-world attack scenarios: (1) conditional password guessing, which recovers the complete password given a partial password; (2) targeted password guessing, which compromises the password(s) of a specific user using their personal information; (3) adaptive rule-based password guessing, which selects adaptive mangling rules for a word (i.e., base password) to generate rule-transformed password candidates. The experimental results show that our fine-tuned models can outperform the state-of-the-art models by 14.53%, 21.82% and 4.86% in the three attacks, respectively, demonstrating the effectiveness of bi-directional transformers on downstream guessing attacks. Finally, we propose a *hybrid password strength meter* to mitigate the risks from the three attacks.

1 Introduction

Textual passwords remain a dominant access control mechanism in the foreseeable future due to their sound usability [5, 6, 61]. Accompanying passwords' ubiquity, decades of research have been conducted on password guessing attacks [20, 26, 32, 57], where data-driven models (e.g., Markov [28, 32]) and rule-based tools (e.g., Hashcat [17]) were used to efficiently crack passwords offline.

While most research focuses on ***general guessing attacks*** with no prior information on target passwords, hackers often collect extra scenario-specific knowledge (e.g., personal information) to further expand attack opportunities (e.g., side-channel attacks [29, 63]). Such attacks, referred to as ***real-world guessing attacks*** in this paper, are increasingly common. Typical examples include targeted password guessing (TPG) [10, 35, 45, 55], which compromises password(s) of a specific user using his/her personal information. Due to the prevalence of personal information leakage, TPG is becoming an increasing security concern in public.

Other real-world guessing attacks like conditional password guessing (CPG) and adaptive rule-based password guessing (ARPG) have also started receiving attention. In 2021, Pasquini et al. [37, 38] proposed two models for CPG and ARPG. Here, CPG recovers complete passwords given a partial password (e.g., "$p***w0rd***$"). CPG is practical when attackers somehow collect a partial password (through malicious monitoring in a surveillance camera or shoulder-surfing). ARPG refers to automatically select adaptive rules for a word (i.e., base password) to generate rule-transformed password candidates. These real-world guessing attacks also threaten password-based authentications.

Bi-directional transformers have received significant attention in the natural language processing (NLP) field [12, 27, 51]. Due to the ability to capture bi-directional context information and high transferability [51], transformers have been effective at multiple language tasks (e.g., text classifications [59], grammatical correction [34]). All password guessing attacks, whether they are general or real-world-based, can be conceptualized as efforts to approximate the probability distribution of passwords (i.e., texts), suggesting a natural fit for a bi-directional-transformer-based guessing framework.

Effectively transferring bi-directional transformers to password guessing attacks is still a considerable challenge. Prior works trivially applied transformers from NLP to general guessing attacks [22] and to ARPG [37], none were able to outperform their state-of-the-art counterparts. This shows that successfully applying transformers to guessing attacks is not

straightforward, and case-specific design is required. As an example, while existing TPG model (i.e., the state-of-the-art *Pass2Path* [35]) utilizes the *sequence-to-sequence* mechanism, our design employs the *sequence labeling* mechanism, in which a categorical label is assigned to each element of the text sequence.

In this paper, we propose a character-level bi-directional-transformer-based guessing framework, referred to as PassBERT, which applies the paradigm of *pre-training* and *fine-tuning* to real-world password guessing attacks. First, we pre-train a general password model using the unlabeled passwords. Then, we design attack-specific fine-tuning approaches to tailor the pre-trained password model to three attack models of CPG, TPG and ARPG. The fine-tuning approaches generally involve modifying model architecture to fit the attack-specific input and output/label format, and re-training the model with the respective objective functions.

Our experimental results show that our fine-tuned models can outperform the state-of-the-art models by an average of 14.53%, 21.82% and 4.86% in CPG, TPG and ARPG attacks, respectively, demonstrating the effectiveness of bi-directional transformers on downstream password guessing attacks. Further, we also analyze the effect of pre-training with ablation experiments, where the attack model is initialized with the pre-trained natural language model or random variables. We find that both pre-trained models (trained on either passwords or natural language) can provide notable gains in untargeted attack scenarios (i.e., CPG and ARPG) with the aid of priori knowledge, while exhibiting marginal gains in targeted attacks (i.e., TPG) due to the task-relevant objective. We also find that, in general, the pre-trained password model can yield better guessing performance than the pre-trained natural language model, showcasing the effectiveness of pre-training on password-specific corpus.

We also introduce a *hybrid password strength meter (HPSM)* (open sourced [1]) with sub-second latency to mitigate risks from CPG, TPG and ARPG attacks. With HPSM, we show each character's strength so that users can modify the vulnerable one(s) with more security gains. Also, HPSM alerts users to the risks of targeted guessing attacks when their passwords can be cracked by our TPG model in small number of guesses. Furthermore, HPSM highlights the base words for the input password (e.g., given an input of "p@ssw0rd123", the base word of "p@ssw0rd" can be inferred). HPSM can be combined with the password leakage checkup [23, 47] to detect whether the input is publicly leaked, and once leaked, then the input suffers from ARPG attacks.

We summarize our main contributions as follows:

- We propose a bi-directional-transformer-based guessing framework, which uses the *pre-training* and *fine-tuning* paradigm. We demonstrate the effectiveness of the pre-trained password model and share it to the community.
- With our framework, we design three attack-specific fine-tuning models for CPG, TPG and ARPG, all of which outperform the state-of-the-art models.
- We introduce a *hybrid password strength meter (HPSM)* with sub-second latency to mitigate these risks.

[1] https://github.com/snow0011/PassBertStrengthMeter.

2 Background and Related Works

2.1 Password Guessing Attacks

General password guessing attacks. Password guessing attacks can date back to 1979 [31, 33, 61], when brute force exhaustion and dictionary-based attacks have been proposed. Later on, researchers presented several guessing attacks, which are mainly divided into data-driven models [30, 32, 57] and rule-based guessing tools [17, 26], to effectively crack generic passwords within larger guesses (e.g., 10^{14}) in offline scenarios [53, 54, 60]. While online guessing assumes that service providers may take protective measures to limit the number of attacks, and consequently, the goal is to crack passwords within smaller guesses. Most of the online guessing works are therefore doing targeted guessing [10, 24, 55].

Data-driven guessing models generally train the probabilistic models based on the observed passwords to enable an educated exploration of candidate passwords. Many model designs have been developed or adapted for this purpose, including Markov [28, 32], Probabilistic Context-free Grammars (PCFG) [24, 52, 56, 57, 60], neural-network-based models (FLA) [30] and generative adversarial networks (GAN) [39]. These models usually aim to estimate the strength via the effort of cracking (i.e., the number of guesses) [11]. Rule-based guessing tools are another threat available in hacking/pentesting tools like Hashcat [17, 26, 62]. As the name suggests, rule-based tools apply the mangling rules (e.g., "delete the last three character") to a word to produce rule-transformed password candidates. Rule-based tools [17, 26] are widely used by actual hackers, and are highly sensitive to their initial configurations [3, 21, 26] (e.g., the order of rules), which are usually customized by the expert knowledge.

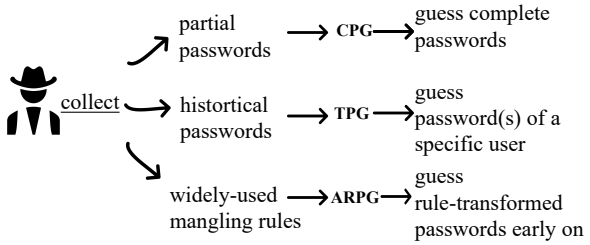


Figure 1: Three real-world attack examples, where hackers collect attack-relevant information to launch attacks.

Real-world attack variants with extra information. Real-world attackers rarely fall into the scenario described in standard guessing developed in academia, while hackers usually collect attack-relevant information to launch pragmatic at-

tacks [35, 37, 38] (e.g., side-channel attacks [29, 63]). These attacks are usually practical in real-world scenarios, narrowing down the search space and reducing economic costs [4]. We show three real-world attack examples in Figure 1, and illustrate them as follows.

CPG [38, 39] cracks complete passwords given a partial password (like "$p***w0rd***$" or "$rockyou******$"). When hackers can somehow get a partial password (through malicious recording in a surveillance camera or shoulder-surfing) or might want to crack a password containing a website name or other sub-string, they can launch the pragmatic attacks of CPG. Furthermore, users can leverage CPG to evaluate each character's impact to final security. A simple solution for CPG is to apply data-driven models (e.g., Markov) to produce a large number of candidate passwords, and then filter out those noise passwords, which is inefficient and storage-demanding. In 2021, Pasquini et al [38, 39] proposed a state-of-the-art approach for CPG based on Wasserstein Autoencoder [48]. They refer to the final model as *CWAE* (*Context Wasserstein Autoencoder*) that consists of an encoder embedding a partial password into latent representations, and a decoder converting the latent representations of the partial password into passwords.

TPG [35, 55] describes that attackers have collected personal information (e.g., historical passwords) to compromise password(s) of a specific user. Given the unending password breaches with Emails [7, 9], the limit of failed login attempts, and the estimated higher reuse rate (e.g., $15-60\%$) across users' passwords [45], TPG attacks are becoming a growing security concern. Besides, a legitimate user can be interested in recovering his/her forgotten passwords. Das et al. [10] proposed the first academic work on targeted attacks. Subsequent works [24,35,55] investigated several improved targeted guessing approaches like *Personal PCFG* [24], *TarGuess I* $\sim$ *IV* [55] and *Pass2path* [35]. Among them, *Pass2path*, proposed by Pal et al. [35] in 2019, is the latest and most effective targeted guessing model. They proposed *credential tweaking* that cracks variants (tweaks) of a user's historical password.

ARPG [37] illustrates that attackers automatically select adaptive rules for each word in a dictionary, where each word only associates to the selected (i.e., adaptive) rules to produce rule-transformed password candidates. The mangling rules in rule-based guessing tools are not necessarily effective and have a conditional nature that should be accounted to seek optimal configurations. The adaptive rules are expected to interact well with the given word. Consequently, ARPG aims to hit the target passwords early on compared with the standard rule-based guessing tools. Further, ARPG can reduce the bias of configuring the order of rules by experts. Attackers can collect widely-used rules to build a model capturing the adaptive relationship between words and rules. In 2021, Pasquini et al. [37] proposed the first adaptive rule-based guessing framework called *ADaMs* (*Adaptive Dynamic Mangling Rules Attack*), which builds a CNN (convolutional neural networks) model to select adaptive rules for each word.

Baseline attack models. We use the state-of-the-art models of *CWAE*, *Pass2path* and *ADaMs* as our baseline for CPG, TPG and ARPG attacks, respectively, since these models has the highest guessing performance with a full comparison with other models. For example, *CWAE* has been proven to be better than PCFG, Markov and neural-network-based models. We therefore do not check comparison with other models.

2.2 Bi-directional Transformers

Bi-directional transformers are first proposed by Google Brain [51] based on the self-attention mechanisms (i.e., connecting the given token with all textual environment), gaining popularity in NLP community due to the ability of capturing deep text features. BERT (Bi-directional Encoder Representation from Transformers) [12, 27] is a popular transformer-based architecture, and has achieved the state-of-the-art results on 11 individual NLP tasks. We consider to extend the BERT architecture to efficiently improve the password guessing tasks, because BERT can well capture bi-directional representations. Although many works improved BERT (e.g., RoBERTa [27] pre-trained upon more training texts; UniLM [14] combined the GPT model [40]), these variant architectures seem not to make a huge difference in capturing bi-directional text features.

BERT is pre-trained on two objectives, which are MLM (Masked Language Modeling) and NSP (Next Sentence Predictions), to build a pre-trained language model based on a large amount of unlabeled web corpus. For the MLM objective, BERT trains the model such that it should be able to predict the correct tokens at the masked positions. Due to the natural-language characteristics, BERT is in large part designed to predict the masked words. The NSP objective is to take a sentence pair A and B, and to predict whether B is the actual next sentence that comes after A. BERT [12] presented two secondary-training approaches: *fine-tuning* and *feature-based*. In the *fine-tuning* approach, all parameters are updated during the downsteam tasks. While in the *feature-based* approach, fixed features are extracted from the pre-trained parameters by freezing some general pre-trained layers. Generally, the *fine-tuning* approach yields better results with more training time [12]. In this paper, we choose the *fine-tuning* approach that all parameters are learnable.

3 Preliminaries

3.1 Workflow of Three Attacks

Supervised learning. Supervised learning is a widely-used machine learning task defined by its use of supervision signals (i.e., the labeled sets of input-output samples) to learn a mapping relationship from the input to its output. We denote the supervision signals as a set of data X with its correspond-

ing set of labels Y, and then train a supervised model with the objective of mapping each $x \in X$ to its label/class $y \in Y$. During training, the model gradually updates its parameters (i.e., weights) so that its output becomes as close as possible to the label y given an instance x. This is achieved by minimizing loss that measures the distance between the predicted and expected output, where we mainly use the cross-entropy loss function.

In this paper, we empirically evaluate three real-world guessing attacks of CPG, TPG and ARPG. Generally, the workflow of these attack models is to train a supervised model based on the supervision signals. We summarize their supervision signals in Table 1. Specifically, the supervision signals of CPG are partial passwords with their complete passwords. CPG aims to train the model such that it should predict the correct passwords given a partial password. The outputs of CPG directly serve as the password candidates.

The supervision signals of TPG are passwords with the shorest edit paths calculated by algorithm of dynamic programming (implemented in [35]). The edit path is a sequence of atom edit operations (pre-defined in our attack designs) like *(delete,8), (delete, 9), (delete, 10)* that can transform a password to its variants. Given a leaked password, the TPG model can output multiple edit paths with varying confidence. We then apply the edit paths to the leaked password to obtain its variants as password candidates, which are used by attackers to compromise other passwords from the same user.

The supervision signals of ARPG are words with the hit rules (i.e., a subset of rules-set) like *delete the last three characters* based on the hit information between two hypothetical datasets. The ARPG model outputs adaptive rules, which are in turn applied to the word to obtain the password candidates. The adaptive rules are usually more compatible to the word, making it possible to produce hits early on. The mangling rules used in ARPG are defined in Hashcat. Generally, most of mangling rules can be a combination of common atom rules (e.g., "deleting last three characters" is a combination of three atom rules of "deleting last characters"), and can naturally simulate the scenarios of applying more than one rules sequentially to the base word.

3.2 Threat Model

In our study, we primarily model the case of *real-world guessing* attacks [35, 37, 38], launched widely by hackers, aiming to maximize the guessed passwords given a limited budget of guesses. Attackers collect attack-specific information (e.g., partial passwords, historical passwords, or widely-used mangling rules), and usually resort to supervised learning to learn the function between a set of inputs and an associated set of outputs. We assume that attackers can choose pre-trained natural-language and password-specific parameters (as a priori knowledge) or random variables to initialize their attack models.

3.3 Password Breach Datasets

To date, a large scale of breach accidents has leaked user credentials, which are publicly available online. We select several datasets used in prior works [16, 21, 37, 38, 43, 54, 60] in our experiments. The datasets used for targeted guessing have Emails, whereas datasets used for untargeted guessing attacks are plain-text passwords.

The password pre-training, CPG and ARPG use the datasets consisting of plain-text passwords as follows.

• **`Rockyou-2009`, `000Webhost`, `Neopets`, `Cit0day`, `Rockyou-2021`:** These datasets are all with English users and widely-used in various works. `Rockyou-2009` [41] is an old dataset including around 32 million passwords from the gaming-related Rockyou websites leaked in 2009. Both `000Webhost` [2] and `Neopets` [1] are leaked in 2016 from the respective two websites. The 000Webhost website supports free websites hosting solutions, and the Neopets website provides pet information. `Cit0day` [8] is a newer data breach including up to 226 million usernames and passwords leaked recently in 2020. Here, we remove the username information in `Cit0day`. The `Rockyou-2021` and `Rockyou-2009` [42] are different datasets, where `Rockyou-2021` represents the latest and biggest datasets from the breach happened in 2021.

For TPG attacks, we select the following two datasets containing Emails and summarize basic information in Table 2.

• **`BreachCompilation (4iQ)`**: The data breaches include a collection of several well-known websites of LinkedIn, Yahoo, MySpace, Twitter, Neopets, etc. The data breaches were first reported by 4iQ in 2017 [7].

• **`Collection#1`**: The data breaches are a credential database containing delimiter-separated Emails with the corresponding passwords that are leaked in 2019 [9].

Joining accounts. To find the password list belonging to the same user, we merge the accounts (users) based on the same Email address [35]. This heuristic strategy would merge several passwords belonging to the same user in most cases, since the same user usually registers with the same Emails. We calculate the statistics in Table 2, where we find most of users (above 97.1%) have no more than 10 passwords. The datasets' reuse rate (i.e., the user adopts the same passwords across multiple websites) is much lower than the reported results [10,45] (i.e., around $15-60\%$). The reason could be that hackers have already removed duplicate passwords from the same user as claimed in [35]. Note that this would not impact our evaluation, because, for the purpose of evaluating targeted guessing attacks, we focus on compromising non-duplicate passwords from the same user.

Dataset cleaning. We adopt commonly used cleanup strategies [16, 35, 37, 60] to filter out the hashed passwords, non-ASCII passwords and abnormally long passwords with more than 32 characters in original datasets.

Table 1: Summary of labeled datasets used in three real-world attacks, where all attack models' objective is to predict its labels (Y) given the input (X) as accurately as possible.

Attacks	Labeled sets a set of data X with its set of *labels* Y	Example $(x \longrightarrow y)$ with $(x \in X, y \in Y)$	Explanations of labels Y
CPG	partial passwords with *complete passwords*	p***w0rd*** $\longrightarrow$ *p@ssw0rd123*	complete passwords are those matching partial contexts
TPG	passwords with *minimal edit paths*	p@ssw0rd123 $\longrightarrow$ [(*delete, 8*), (*delete, 9*), (*delete, 10*)]; We can change it to "p@ssw0rd", where the [(*delete,8*)] refers to delete the character 1 in the eight position.	the minimal edit path (i.e., the shortest sequence of edit operations) refers to the shortest path to change the current password to another password from the same user.
ARPG	words with *hit rules*	p@ssw0rd123 $\longrightarrow$ [*delete last three characters*, *duplicate the last character once*]	hit rules are a subset of rules-set that is preprocessed the word and rule mappings upon two hypothetical datasets

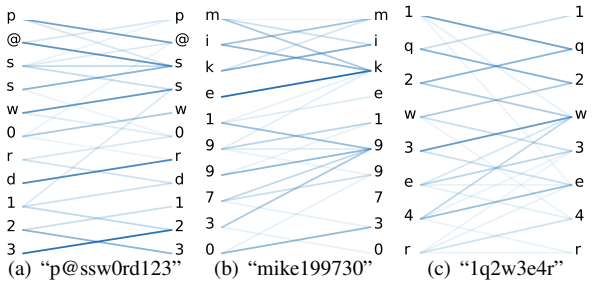
Table 2: Summary of password datasets with personal information, whose raw format contains Emails with the corresponding several passwords.

		BreachCompilation (4iQ)	Collection#1
Accounts / Users		147,284,401	109,191,685
Passwords		373,820,141	365,336,365
Passwords per user	≤ 10	99.3%	97.1%
	>10	0.7%	2.9%
Password reuse rate		4.2%	0
Edit distance	≤ 4	21.8%	28.1%
	>4	78.2%	71.9%

Ethical claim. Our work only presents the statistical information for the requirement of ethical practice. While we use real-world datasets that include Emails, we do not identify the exact user of the leaked passwords. Instead, we focus on the whole feature collection of many user's passwords in a breached dataset. Further, we believe that our work is ethical based on the following features in the literature [46]: (1) *public data* (i.e., we only use the publicly available datasets and do not share them with others). (2) *necessary data* (i.e., this research cannot be conducted without these datasets). (3) *no additional harm* (i.e., our research does not identify any personal information with data being managed [2] securely).

3.4 Password Bi-directionality

As a text, the password also exhibits bi-directionality. As shown in Figure 2, we visualize the bi-directionality by showing the self-attentions of a character with its context. The colors ranging from light to dark correspond to the connections from weak to strong. Different from the uni-directional representations that each character only associates with the previous characters, we find that characters connect differently with other characters given bi-directional contexts. We conclude the following password bi-directional characteristics: **Sequentiality:** characters are generally more relevant with their adjacent characters, which can also be a manifestation of uni-directionality. **Aggregation:** The inner sequence of relevant characters (e.g., "p@ssw0rd" and "123" in "password123"; "199730" in "mike199730") has more connected lines. Our hypothesis is that capturing bi-directional representation can enable better password candidates, boosting password guessing efficiencies.



(a) "p@ssw0rd123" (b) "mike199730" (c) "1q2w3e4r"

Figure 2: Password bi-directionality: every character connects other characters with different weights (colors ranging from light to dark correspond to the weak and strong weighted connections).

4 PassBERT Guessing Framework

In this section, we describe PassBERT guessing framework, which applies the paradigm of combining *pre-training* and *fine-tuning* to password guessing attacks.

4.1 Password Pre-training

We mainly present the password-specific design decisions that are necessary to transfer transformers to password modeling, ranging from the model architectures and the pre-training processes.

Pre-trained model architectures. We tokenize a password as a sequence of characters (in the bottom part of Figure 3) with additional symbols denoting the beginning ([CLS]) and ending ([SEP]), since passwords are usually shorter than sentences and without a preset common dictionary like the words in natural language. Unlike BERT, which generally tokenizes a text in *token-level* whose tokens are primarily words and

[2] Although the leak is publicly available on Internet, we do not want to publicize it and process the datasets by a computer not connected to internet.

clips each sentence to 512 tokens. We consider the maximum password length to 32 characters, and consider a total of 99 valid characters including 95 ASCII characters (denoted as $\sum$) and 4 additional symbols of starting, ending, placeholder, and unknown characters.

The embedding layer would convert the tokenized input into its input embedding (i.e., high-dimensional representations without contextual information) by summing up its character and position embedding (in the middle part of Figure 3). We remove the sentence embedding in BERT. We show the pre-trained password model architectures in Table 9 (in Appendix B).

Password pre-training process. Multiple transformer blocks process the input embedding for feature extraction with the only objective of MLM to train the pre-trained password model (in the top part of Figure 3). We remove the NSP objective since there is nothing equivalent to the next sentence in the password domain.

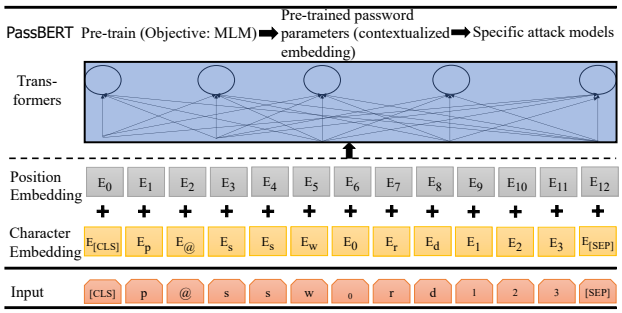


Figure 3: Overview of PassBERT.

To pre-train the password model with the MLM objective (i.e., predict the masked characters behind the masked positions), we pre-process each password in the training sets ($D_{training}$) to the form of the partial passwords (denoted as *pivot*) with the associate complete passwords (denoted as *pwd*).

We follow the same masking proportions in BERT [12]: we randomly choose 15% of the characters in a password and then replace the selected characters with the masked symbols, random characters and unchanged characters with 80%, 10%, and 10% probability, respectively. The random and unchanged characters can prevent the model remembering the masked characters. A password can be pre-processed to many pivots, of which we set up 20 pivots in this paper. We set the pre-training task that finds the parameters θ to maximize the following likelihood:

$$\underset{\theta}{argmax} \quad \frac{1}{|D_{training}|} \sum_{(pivot,pwd)\in D_{training}} \log P(pivot \rightarrow pwd|\theta)$$

Pre-training datasets. We use the `Rockyou-2021` as our pre-training dataset, which is extremely large. To strike a balance between training time and model performance, we randomly sample 60 million passwords from `Rockyou-2021`.

Computational performance. Our work is performed on one Ubuntu 20.04 machine equipped with Nvidia GeForce RTX 2080 Ti, and takes around 2 days to complete pre-training. It takes 8.9 MB to store the pre-trained model.

4.2 Password Fine-tuning

Through a dedicated design of the task-specific layers and objective function, we can tailor the pre-trained model to specific attack scenarios. Our fine-tuning approach generally includes architecture modifications and model re-training. During architecture modification, we usually modify task-specific layers (the fully connected layer and the output layer in Table 9) of the pre-trained model architecture, and keep the pre-trained layers including the top input layer, the embedding layer and several transformer blocks. Then, we re-train the downstream models with specific supervision signals to learn the task-specific features (e.g., password-rule compatibility). Note that all parameters (of the pre-trained layers and the task-specific layers) are updated in our fine-tuning approach.

The pre-trained model mainly captures the contextualized embedding of an input password, which is the high-dimensional representations with contextual information for each character in a password. The contextualized embedding is the output of the pre-trained layers (i.e., the last transformer blocks). To illustrate, the contextualized embedding of "s" in "p@**s**sw0rd123" and "te**s**t123456" is different given the varying contexts, although both "s" share the same input embedding. Then, the contextualized embedding of the same input is transferred to the task-specific layers and can align with the downstream models. When we train the specific models with pre-training, the pre-trained layers of specific models are initialized with parameters of the contextualized embedding from the pre-trained models. When we train the specific models without pre-training, the pre-trained layers are initialized with random variables.

5 PassBERT for Real-world Attack Models

In this section, we present the fine-tuning designs of three real-world attacks along with their empirical evaluation.

We make ablation experiments that train the attack model in the same way, except that the attack model is initialized with the pre-trained parameters from natural language and random variables. We use the term PassBERT, Vanilla BERT and *PassBERT to refer to the approaches, where we train the attack model with the pre-trained password model, the pre-trained natural language model of BERT [3], and random

[3] We choose the open-sourced pre-trained BERT model (https://github.com/google-research/bert) with same number of transformer layers of PassBERT. Specifically, we employ the BERT-Mini with 4 transformer blocks of 256 dimensions.

variables. Vanilla BERT can tokenize a text (i.e., password) in *word-level*, *subword-level* and *character-level*, while we only tokenize a password as a sequence of characters based on the vocabulary. As the vocabulary of our Vanilla BERT only has the lowercase letters, we expand the vocabulary to cover all valid characters in $\sum$ (via replacing the unused tokens).

5.1 Conditional Password Guessing

CPG recovers passwords given a partial password, which is denoted as a **pivot** (e.g., "$p***w0rd***$"). We model CPG as a masked language model task (e.g., cloze test [13, 27]) that aims to predict the missing characters' conditional probabilities of a pivot. The pivot in CPG is created with the same strategies as *CWAE* [38, 39] as follows.

We randomly replace each character at a certain percentage (i.e., 50%) with a masked symbol denoting the missing characters. Then, we keep only those of the produced pivots containing at least four observable characters and at least five masked symbols. These constraints guarantee that brute force has a search space of around 7.7×10^9 ($|\sum|^5, |\sum| = 95$).

5.1.1 Fine-tuning

Attack design. We mainly modify the masking mechanisms based on the consistency of CPG pivot-creation policies for model training as follows:

We increase the masking proportion from 15% to a larger value (i.e., 50%) in a password and always replace the selected character position with a masked symbol, because CPG has the only objective of predicting the correct characters behind the masked symbols.

We also try the default masking mechanisms of the pre-trained model, while yielding unsatisfactory results. Then we adapt the masking mechanisms to fit the CPG used case, which enables transformers to outperform *CWAE*.

Model architecture. The model architecture is shown in Table 10 (in Appendix B), where CPG keeps the task-specific layers as the pre-trained model. With the CPG-specific masking mechanisms, we re-train the model to predict the correct passwords given a pivot.

Model re-training. Same as *CWAE*, we re-train the CPG model using `Rockyou-2009`, from which we extract valid pivots with the correct passwords for model re-training. We denote $mask_i$ as the i-th masked position, c_i as the characters behind the $mask_i$. Then CPG predicts the probability of the correct password (*pwds*) given a *pivot*:

$$P(pwd \mid pivot) = \prod_{c_i \in pwd,\ mask_i \in pivot} P(c_i \mid mask_i, pivot)$$

We supplement the hyper-parameters of the model re-training in Table 13 (in Appendix B).

5.1.2 Evaluation

Experimental settings. We infer our CPG models and the open-sourced *CWAE* model [4] with the same evaluation pivots. Same as *CWAE*, we generate 10^7 candidate passwords per evaluation pivot, and classify the evaluation pivots into four classes by the number of passwords satisfying the pivot (N_{pivots}): (1) *common* if $N_{pivots} \in [1000, 1500]$; (2) *uncommon* if $N_{pivots} \in [50, 150]$; (3) *rare* if $N_{pivots} \in [10, 15]$; (4) *super-rare* if $N_{pivots} \in [1, 5]$. These four classes can better differentiate the guessing performance of pivots with varying frequencies. Samples of evaluation pivots and their respective cracked passwords by PassBERT are showed in Table 4.

We use `Neopets` and `Cit0day` as evaluation sets due to their large space. *CWAE* only generates 30 evaluation pivots for each pivot class, possibly inducing bias. We extract a total of 120 evaluation pivots for each class from evaluation sets for robust and convincing results.

The overlap ratio (i.e., the pivots and labeled passwords are the same between training and evaluation datasets) is small. We empirically count that only 0.27% (0.01%), 0.14% (0.01%), 0.27% (0.01%), 0% (0%) in respective *common*, *uncommon*, *rare* and *super-rare* pivot class, between fine-tuning (pre-training) datasets and evaluation sets of `Cit0day`. We believe that such a small overlap is neglectable that cannot bias the conclusion. Another evidence is that our models achieve significant improvement in *super-rare* pivots without overlap.

Evaluation metrics. The evaluation metrics is the average cracking rates among 120 evaluation pivots in each pivot class. The cracking rate for each evaluation pivot is $\frac{N_{intersection}}{N_{pivots}}$, where the $N_{intersection}$ is the intersection between the 10^7 candidate passwords and the passwords satisfying the pivot in a evaluation set.

Experimental results. As shown in Table 3, our three CPG models outperform than *CWAE*. PassBERT maintains the highest guessing performance. Regarding the four pivot classes together, PassBERT improves cracking rates by 18.53% (from 61.64% to 73.06%) in `Cit0day`. Similarly, PassBERT improves by 10.54% in `Neopets`. We calculate that PassBERT improves by an average of 14.53% than *CWAE* based on the two evaluation sets. We also find that the cracking rates are higher as the N_{pivots} increases (from *super-rare* to *common*), indicating that our models can produce more confident passwords given more common pivots.

Pre-training effect. We find that both the pre-trained models (trained on either password or natural language provide notable gains, where our pre-trained password model has a higher performance, demonstrating the effectiveness of pre-training upon password-specific corpus in CPG. Moreover, pre-training can be more beneficial when pivots become scarcer. E.g., PassBERT achieves more improvements than *PassBERT in *super-rare* pivots. This is because the N_{pivots}

[4] https://github.com/pasquini-dario/PLR.

Table 3: Cracking rates of CPG. *CE*, **PT* *VT* and *PT* refers to *CWAE*, **PT*, Vanilla BERT and PassBERT, respectively.

pivots	Neopets (%)				Cit0day (%)			
	CE	**PT*	*VT*	*PT*	*CE*	**PT*	*VT*	*PT*
common	68.62	74.04	77.25	**80.02**	67.65	75.66	79.90	**83.23**
uncommon	77.35	73.88	79.40	**83.51**	69.30	72.80	76.18	**80.06**
rare	70.62	75.52	76.07	**79.72**	63.70	70.08	71.83	**76.48**
super-rare	69.86	59.51	62.25	**73.41**	45.90	46.11	47.86	**52.50**
average	71.61	70.73	73.74	**79.16**	61.64	66.16	68.94	**73.06**

of such pivots in the training sets is small, making it harder for model convergence with only random parameters. The scarce pivot cases (e.g., *super-rare* pivots) can be widely encountered by hackers, e.g., when these hackers aim to compromise the pivots with many customized restrictions (e.g., the domestic or personal composition habits).

Improvement principles. Our models generate candidate passwords by exhausting characters in a masked position with descending probabilities, while *CWAE* translates a latent representation (i.e., a point in high-dimensional space) to a candidate password via decoders. Take "1997$*****$" as an example. *CWAE* generates noise passwords (e.g., "1565s19i" or "79571deS") that are not satisfied with the pivot within small guesses, while our PassBERT does not generate noise passwords (shown in Table 4). Still, we can beat the guessing performance even when removing the noise passwords generated by *CWAE* (in Table 3).

Computational performance comparison. It takes around 2 days to train our CPG model (8.9 MB), and takes around a day and a half to train *CWAE* (4.4 MB). Given an evaluation pivot, we can infer candidate passwords much faster than *CWAE*. We empirically calculate that the average inference speed is 4.54 and 0.08 passwords per second (pwds/s), respectively in PassBERT and *CWAE*. The reason behind this phenomenon is that PassBERT can generate all password candidates based on the inferred conditional probabilities with a single inference, whereas *CWAE* can only produce one candidate password during each inference because *CWAE* needs to go through the decoder network to every output password.

5.2 Targeted Password Guessing

TPG aims to compromise password(s) of a specific user using historical passwords. Specifically, Given a historical password, TPG generates its variants. We train the TPG model to output the edit paths given a leaked password. Here, the produced edit paths are in turn applied to the input password to produce candidate password variants, which are used by attackers to compromise the other passwords from the same user.

5.2.1 Fine-tuning

Attack design. The mainstream TPG model, *Pass2path* [35], transforms a password to an edit path based on the *sequence-to-sequence* mechanism of RNN model. We cannot simply replace the RNN model with our transformer encoders, since encoders do not support such mechanism. Instead, we use the *sequence labeling* mechanism [44] that predicts one edit operation for each character in a password, where a sequence of edit operations makes the edit path. In the *sequence labeling* mechanism, each character position can only output one edit operation.

To adapt the *sequence labeling* mechanism, we pre-define our new edit operations as follows:

keep *(keep)*, ***delete*** **(***del***)**, and ***replace*** **(***rep*$_1$**,** *rep*$_2$**)**

Here, ***replace*** involves replacing with one (denoted as rep_1) or two characters (denoted as rep_2).

Note that we do not adopt the design of *add_before (after)* of one character, because the one-character design cannot cover the transformations with both *replace* and *insert* operations with one character. For example, the widely used transformation case of "*password* $\rightarrow$ 1*Password*" can only be captured in our design of rep_2, since the position of "p" can only output either *replace* or *add_before*.

Formally, we denote an edit operation as a binary-tuple of (op, str), where $op \in \{keep, del, rep_1, rep_2\}$ denotes an operation and $str \in \sum \cup \sum^2 \cup \{EMPTY\}$ denotes a string. *str* is always an empty string (*EMPTY*) for *keep* and *del*, and one or two characters to be replaced with for rep_1 and rep_2. For example, when we separately perform the edit operation of (*keep*, *EMPTY*), (*del*, *EMPTY*), (rep_1,*b*) and (rep_2,*b*!) on the last character of "p@ssw0rd123", we transform it into "p@ssw0rd123", "p@ssw0rd12", "p@ssw0rd12b" and "p@ssw0rd12b!", respectively. As we consider $|\sum|$ as 95, the total of edit operations is 9,122:

$$9122 = 1(keep) + 1(del) + |\sum|(rep_1) + |\sum|^2(rep_2)$$

To simulate the *append* operation, we also add three placeholders in the end, since most of the password variants are appended with no more than three characters in the end.

Limitations. Our design cannot capture some transformations like inserting three characters in a password or two characters before a password like "*password* $\rightarrow$ 12*password*", which are relatively rare in datasets (5.04% in BreachCompilation).

Model architecture. We show the model architecture in Table 11 (in Appendix B), where we change the task-specific layers to learn probabilities of edit operations, e.g.,(op, str), for each character in a password. The output layers are projected with 9,122 edit operations (i.e., our solution's spaces). We take the "p@ssw0rd123" as an example to explain the process of inferring its variants. We first tokenize it as "[CLS] p @ s s w 0 r d 1 2 3 _ _ _ [SEP]" by adding the starting, ending symbol and three placeholders of "_" in the end. Then, our TPG model outputs the top edit path in the form of "[CLS]

Table 4: Samples of pivots along with their respective top-5 cracked passwords by PassBERT in Neopets. E.g., we can recover 99.3% of the correct passwords given the pivot of "*e*sica***" in 10^7 guesses, in which the "jessical11" is the most possible (top-1) password, carrying semantics of a common female name in English.

	common			*uncommon*			*rare*			*super-rare*		
	1997*****	*e*sica***	b*****101	be*i*****7	j*ro****5	12c*****e	ke***ten***	**p**mu**2	**ke**91**	#***i***ir*	**2*k**u*a	***l***s*y22
Rank	51.5	99.3	94.5	60.9	92.9	97.3	60.0	81.8	93.3	20.0	50.0	75.0
1	19971031a	**jessical11**	barbee101	belinea007	jerome415	12charlie	kerenteng11	supermua12	mike199130	#1chinagirl	1029kyouka	hellonasty22
2	199710303	jessical12	babbie101	benita1107	jerome405	12cherrie	kevinteng12	septimus12	mike109135			tillysassy22
3	19971230a	jessicali1	benben101	betina2007	jerome245	12cherise	keirsten101	septimus22	nike299100			hollycasey22
4	199703100	jessicale1	barbey101	benito2007	jerome815	12charrie	keirsten123	septimus42	mike159157			
5	199702313	jessical10	bobben101	belinha007	jerome345	12chelsie	keersten123	septimus82	mike999175			

1 1 1 1 1 1 1 1 10 1 1 9121 9122 _ [SEP]", where each numeric value indicates an edit operation (op, str) from 9,122 operations. We then obtain the probability of edit paths (i.e., the probability of variants) by multiplying the edit operations' probabilities.

Model re-training. Following *Pass2path* [35], we randomly sample 80% password pairs from the same user in BreachCompilation and choose the qualifying password pairs when their minimal edit distance is no more than 4, resulting in the total of 85,269,455 password pairs. With the minimal edit operation as edit labels, we aim to predict the probability of (op_i, str_i) for every characters (c_i):

$$P(variant \mid pwd) = P(edit\_labels \mid pwd) = \prod_{c_i \in pwd} P[(op_i, str_i) \mid c_i]$$

5.2.2 Evaluation

Evaluation settings. We re-train *Pass2path* [35] based on the open-sourced codes [5] to guarantee the same training sets for *Pass2path* and our TPG model. Note that we do not directly use the open-sourced *Pass2path* model due to the conflict of evaluation sets with training sets of *Pass2path*, which randomly samples 80% quantifying password pairs. Furthermore, for evaluation, we randomly sample $N_{accounts}$ (10^5 in our settings) users from the rest 20% of BreachCompilation and Collection#1, where each user has two passwords.

Evaluation metrics. We pick one of the leaked passwords from a specific user as the input, and then infer its variants (limited to 1,000 guesses) based on the TPG models. Once the produced variants hit the other password of the same user, the user's account is cracked. We calculate the cracking rate among the evaluation uses/account by $\frac{N_{cracked}}{N_{accounts}}$ given different guesses (i.e., 10, 100, and 1,000).

Experimental results. As shown in Table 5, our three models significantly outperform *Pass2path*. PassBERT can improve the cracking rates by an average of 21.82% (22.09% in BreachCompilation and 21.56% in Collection#1), demonstrating the transformers' superiority in targeted guessing. We note that the literature [35] of *Pass2path* reported slightly higher results in BreachCompilation (i.e., 9.9%, 13.1% and 15.8% in respective 10, 100 and 1,000 guesses). This phenomenon is possibly due to the larger size of training password pairs in *Pass2path*, which adopts the keyboard-sequence representation. For example, the case of "QWERTY → Qwerty" has five edit distances, which is removed for our model training. While *Pass2path* regards the pairs with two edit distances by the transformed pair of "<c>qwerty<c> → <s>qwerty" (<c> and <s> refer to capitalization and shift).

[5] https://github.com/Bijeeta/cretweak.

Table 5: Cracking rates of TPG.

Attack model	BreachCompilation (%)			Collection#1 (%)		
	10	100	1,000	10	100	1,000
Pass2path	6.42	11.52	14.71	4.37	10.84	14.98
*PassBERT	12.63	15.67	17.94	11.21	15.42	18.22
Vanilla BERT	**12.72**	**15.79**	**18.01**	**11.35**	15.45	**18.23**
PassBERT	12.68	15.71	17.96	11.24	**15.47**	18.21

Pre-training effect. The pre-trained password and natural language models exhibit marginal gains in TPG. We also down-sample a quarter of supervision signals (i.e., around 21 million) to repeat experiments, and find that the improvement room of pre-training still remains small, showcasing the weak role played by pre-training. This phenomenon can be understood since the targeted attacks focus on personalized password transformations, which are less relevant with the global password distributions in the pre-trained model. The contextualized embedding produced by both pre-trained models produce little effect in helping understand password transformations. TPG tends to be task-dependent that forms their model parameters mostly from the attack-specific datasets. For example, the parameter size of TPG is around 82 MB, while the parameter size of CPG is just around 8.9 MB. This is because that the output layer in CPG only connects with 95 characters, while that in TPG connects with 9,122 edit operations.

The targeted attacks could benefit a little more from the pre-trained natural language parameters. The reason can be that users create their password variants based on the language habits (e.g., "two → 2"), as we observe that the widely used web-corpus transformations (e.g., "twofast4u → 2fast4u" or "CMC-17-CMC → CMC17CMC") can be easily cracked by Vanilla BERT.

Table 6: Statistics of edit distance distribution by PassBERT within 1,000 guesses. For example, in BreachCompilation, we can crack 12,102 (98.5%) passwords among the total passwords with one edit operation.

Edit distance	BreachCompilation (4iQ)	Collection#1
1	**12,102 (98.5%)**	**122,17 (96.7%)**
2	16,316 (91.7%)	16,257 (88.6%)
3	17,552 (84.8%)	17,792 (81.5%)
4	17,923 (75.4%)	18,175 (72.8%)
>4	39 (0.05%)	39 (0.05%)

Improvement principles. Our TPG model focuses more on local character information than *Pass2path*. Precisely, our binary tuple of edit operations *(op,str)* can reduce the one-dimension search space than that of triple-tuple operations like *(op,str,pos)* used in *Pass2path*, where *pos* refers to the position in a password. Our model architecture has already encoded the position information, making it possible to learn the binary tuple edit operations. Besides, we find that PassBERT can easily crack position-relevant transformation cases like *del* (e.g., "557gpss → 17gpss"), which cannot be cracked by *Pass2path*.

Insights. We find that the cracking rates can be reduced as the edit distance increases (shown in Table 6), and almost all the target passwords with one or two edit distances from the leaked passwords can be easily cracked. When changing passwords to meet requirements of password expiration [45, 55], we alert users not to adopt minor modified variants and should create their variants that are significantly different from a leaked password.

Computational performance comparison. It takes around 13.6 and 42 hours to train PassBERT (82 MB) and *Pass2path* (166 MB). We empirically calculate that both models achieve similar inference speed. PassBERT and *Pass2path* can infer 4.38 and 4.63 password pairs per second (pairs/s).

5.3 Adaptive Rule-based Password Guessing

During ARPG, each word in a dictionary only associates with the selected adaptive mangling rules to generate guesses. [37] has attempted to use default transformers, while achieving no substantial improvement than *ADaMs*. In our work, through the introduction of pre-training, our model is able to yield a superior guessing performance.

5.3.1 Fine-tuning

Attack design. To capture the adaptive relationship between words and rules, we build a classification model to output a continuous value $v \in [0,1]$ measuring the *adaptability* of a rule r_j on a word w_i. The output value v close to 1 indicates that r_j is adaptive on w_i and would possibly produce hits. In contrast, the output value v close to 0 represents that real scenarios would not apply r_j on w_i and possibly leads to failed guesses.

We capture the adaptive relationship between a whole word and a mangling rule in password-level since the mangling rules ultimately apply to a word.

The supervision signals are words with labeled rules (i.e., hit rules) upon two hypothetical datasets. Formally, given a chosen rules-set $\mathcal{R}$, a wordlist-set $\mathcal{W}$ and a hypothetical target set of passwords $\mathcal{T}$, we preprocess a collection of $(w_i, \mathcal{R}_i)$, in which, for each $w_i \in \mathcal{W}$, $\mathcal{R}_i$ is a rule-list, which contains the rules r_j indicating whether each rule is a hit as follows:

$$\mathcal{R}_i = \{r_j : r_j(w_i) \in \mathcal{T}, j = 1,2,...,|\mathcal{R}|\},$$

where $r_j \in \mathcal{R}$ and $r_j(w_i)$ refers to the guess when applying the rule r_j to the base word w_i. Once $r_j(w_i)$ hits the targets, the position j in the list is labeled with "1"; otherwise it is labeled as "0" in $\mathcal{R}_i$. w_i simultaneously participates with multiple binary classes/rules R; The task can be described as multiple binary classification tasks.

$$< P(w, r_1 \in \mathcal{R}), P(w, r_2 \in \mathcal{R}), ..., P(w, r_{|\mathcal{R}|} \in \mathcal{R}) >$$

Model architecture. We show the architecture of ARPG in Table 12 (in Appendix B), where ARPG changes the task-specific layers to infer the adaptive probability between a mangling rule and a base word with focal loss function. ARPG infers the adaptive rules in *password-level*. The output layers of ARPG do not have seq-length (i.e., the number of characters in the password), while CPG and TPG infer the characters and edit operations in *character-level*.

Model re-training. We train two ARPG models for two rules-sets of *PasswordPro* (3,120 rules) and *Generated* (14,278 rules) in Hashcat. We use BreachCompilation with around 200 million target passwords (after removing Emails) as $\mathcal{T}$ to describe the target space, and use 000Webhost with around 10 million unique words as $\mathcal{W}$ to represent the word space. Note that the size of datasets for model training is smaller than *ADaMs*. With the $\mathcal{R}_i$ as labels, we re-train the attack model to obtain the adaptive score (i.e., given a password, the possibility of a rule hitting the target) between the word w and rules in a rules-set $\mathcal{R}$.

The input word w_i is first tokenized in character-level with a starting symbol [CLS] and ending symbol [SEP]. Afterwards, we follow standard classification practices [12] that use [CLS] embedding to represent the entire word w_i, and use the [CLS] embedding in output layers to predict the adaptive probability for each r_j. Here, the proportion of 0 and 1 is extremely unbalanced, i.e., more than 95% labels are 0. We therefore apply the *focal loss* [25,37] with the same parameter strategies as *ADaMs* to focus on the hard labels.

Given a word w as input, our ARPG model outputs a probability value v for each rule r in a chosen rules-set $\mathcal{R}$. We infer the adaptive rules for w when the v is larger than a threshold

(i.e., a real-value ranging from [0,1]), which implicitly determines the size of inferred adaptive rules. As the threshold increases, the selection is strictly limited with highly adaptive rules for the word. When the value of threshold reaches 0, ARPG becomes standard rule-based guessing attacks that unconditionally apply all rules to the word w.

5.3.2 Evaluation

Evaluation settings. We directly compare the publicly available *ADaMs* model [6], which has an optimized trick of dynamic settings that constantly add the cracked passwords to the wordlists. We show the results of both static and dynamic models for completeness.

We set the thresholds such that the adaptive rules selected via our models are less than those by *ADaMs* to reduce the impact of the size of adaptive rules (i.e., applying more rules can lead to higher efficiencies). We focus on the relationship modeling between words and rules, namely, we aim to select more adaptive rules for a word to achieve higher guessing efficiencies under the same guesses. We use the binary search to settle down 19.3% and 19.4% rules for *PasswordPro* and *Generated* with the threshold of 0.100 and 0.150 in PassBERT, while *ADaMs* chooses 20.1% and 19.7% for two rules-set with the threshold of 0.275 and 0.415 [7].

We use the wordlists of `Rockyou-2009` with around 14 million unique words to crack `Neopets` (27 million unique passwords) and `Cit0day` (40 million unique passwords). We also evaluate the scenarios where the overlap between the wordlists and evaluation passwords are removed from the evaluation sets. We find that such scenarios achieve similar results, except that the final cracking rates are lower than the standard rule-based attack in Hashcat.

Evaluation metrics. We use the final cracking rates of models with dynamic strategies as evaluation metrics.

Experimental results. As shown in Figure 4, we can find that both static models and dynamic models can outperform *ADaMs*. We can also conclude as follows. (1) Dynamic PassBERT achieves the highest cracking rates among these models and manages to improve by an average of 4.86% than *ADaMs* across the four experiments. For example, when cracking `Cit0day` using *PasswordPro*, we increase the cracking rate by 6.35% (i.e., from 34.20% to 36.37%). (2) The significant earlier stop of static ARPG is because that each word only associates with the adaptive rules, whose size is fewer than all rules in standard rule-based attacks. The static model can produce similar (i.e., around 80%) hits in early guesses (i.e., the top 20% guesses) compared with the standard attacks, shortening the economic cost (guesses) [4]. Given that we can achieve similar final cracking rates within the top 20% guesses, standard attacks waste approximately large (e.g., 80%) guesses on pursuing only a little performance boost.

Table 7: Top-5 rules used in cracked passwords by PassBERT in ARPG. We list descriptions of specific function in Table 8.

	PasswordPro		*Generated*	
	`Neopets`	`Cit0day`	`Neopets`	`Cit0day`
1	]	]	**$1**	**$1**
2	**$1$2**	]]	$0Z1	’8
3	Z1	Z1	$2	c$1
4	]]	D2	@0Z1	c’8
5	$1$2$3	[	$2Z1	’7$1

Pre-training effect. Consistent with the previous observations in *ADaMs* [37], the default application of transformers (*PassBERT) obtains no substantial improvement than *ADaMs*. We find that password pre-training enables transformers to outperform *ADaMs*, while natural language pre-training seems not work in ARPG. The results demonstrate the effectiveness of password pre-training in ARPG. This could be due to the less relevance between the ARPG’s objective (password-rule compatibility) and the natural language habits.

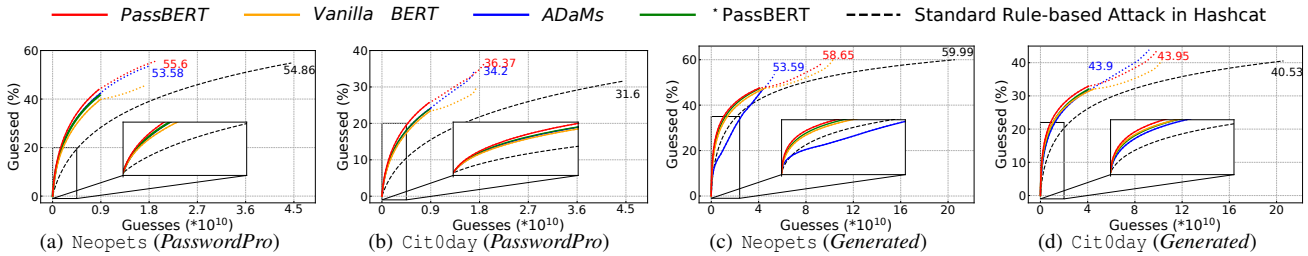
Improvement principles. Password pre-training plays an important role for the improvement of ARPG. ARPG conceptually classifies the adaptive relationship between rules and words, significantly benefiting from the global distributions. The contextualized embedding of a password produced by the pre-trained password model can help understand the specific password structure. This can be analogous to other classification tasks (e.g., emotional classification on a sentence), which must fully understand the whole sentence.

Insights. We uncover the top vulnerable mangling rules to alert their risks. We list the top-5 used rules across the cracked passwords in Table 7, where we show the mangling rules by a sequence of functions in Hashcat. Next, we show the explanation of the function in Table 8 (we can see [18] for complete function explanations). We find that the two mangling rules of “deleting the last character” and “appending the character 1 to the end” are significantly vulnerable, becoming an underestimated risk.

Computational performance comparison. It takes around two hours to train our ARPG model (*Passwordpro*: 46 MB; *Generated*: 115 MB), while it takes around ten hours to train *ADaMs* (*Passwordpro*: 44 MB; *Generated*: 132 MB).We count that both PassBERT and *ADaMs* achieve similar inference speed, inferring similar number of rules’ probabilities per second (r/s) (shown in Table 14 in Appendix C). Although self-attentions in transformers generally need quadratic time, our models are lightweight (i.e., smaller size), resulting in sound latency.

[6] https://github.com/TheAdamProject/adams.

[7] Vanilla BERT chooses 20.7% and 21.7% for respective rules-set; *PassBERT chooses 19.5% and 20.6%, respectively.



(a) Neopets (*PasswordPro*) (b) CitOday (*PasswordPro*) (c) Neopets (*Generated*) (d) CitOday (*Generated*)

Figure 4: Cracking rates of ARPG, where we show the final cracking rates of dynamic models. The dotted line is the result with the dynamic strategies in respective attack models.

Table 8: Explanations of the functions in Table 7. The output is the result when applying the example rule on "password".

Permutation functions	Descriptions	Example	Output
]	Delete the last character	]	passwor
$x	Append the character x to the end	$1	password1
Zn	Duplicate last character n times	Z2	passworddd
@x	Purge all instances of x	@s	paword
'n	Truncate word at position n	'6	passwo

6 Hybrid Password Strength Meters

Password strength meters. Password strength meters (PSMs) [30, 35, 36, 58, 60] show the strength of passwords to help users change a weak password to a strong one, since users' perception of password security is generally error-prone [49]. Recently, massive works [30, 36, 58, 60] proposed explainable meters to provide suggestions of nudging the weak passwords to become strong. Typical explainable meters include zxcvbn [58], CKL_PSM [60], or the meter [36] proposed in 2020, showcasing the weak elements based on one attack model. However, the changed password may also suffer from other attacks. Suppose that CKL_PSM shows that the substring of "123" in "p@ssw0rd123" is a weak element, users are possible to make minor modifications [10, 15, 50] like "p@ssw0rd12b", which still suffers from targeted attacks.

Hybrid password strength meters. These issues give rise to a hybrid password strength meter (HPSM), where the hybrid idea can be a promising direction for PSMs. The literature [19] has already proposed *Hybritus* as a robust measurement for different websites' policies from multiple perspectives. To our knowledge, the hybrid password meter has not been defined before, even though the motivation is similar to [19]. Our HPSM is to mitigate risks from CPG, TPG and ARPG, and can be combined with the existing meters.

Specifically, we apply the CPG model to estimate the single character strength [36], which tells users the impact that a single character contributes to the final security. Users can directionally modify those labeled weak characters with more security gains. To this end, we mask a character in the password each time and estimate their respective strength given the rest of password contexts. Then we label those characters with higher predicted probabilities as the weak one.

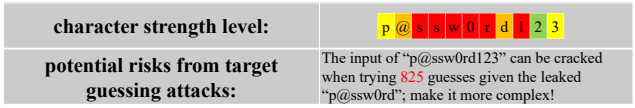


Figure 5: Client-side interface of HPSM. Users can improve their passwords by modifying the labeled red characters with the highest security gains and get hints of TPG attacks.

To alert TPG risks, we use the TPG model to estimate the number of guesses of cracking the input password given the leaked passwords, and alert users when the guesses are smaller (i.e., less than 1,000). The leaked passwords can either from the user's customized passwords entered by themselves or the default settings of the leaked top-20 passwords. [8]

Finally, we apply the reversed ARPG model to infer the base words, e.g., given an input password of "p@ssw0rd123", we can infer the base word of "p@ssw0rd". The base words can be potentially checked via a password leakage checkup [47] in a privacy-preserving manner to detect whether the base words have been leaked. Once the inferred base words are publicly leaked, the input password (no matter how complicated it looks) is vulnerable to ARPG attacks. To infer the base words, we re-train a reverse ARPG model by the reverse supervision signals (detailed in Section 5.3) to capture the adaptive rules from target passwords to base words. For example, we obtain the adaptive rules used to produce the target of "p@ssw0rd123" by "p@ssw0rd". Then, we can apply the inverse rules to the target passwords to obtain the base words. We develop a *rule-inversion* module to convert the rules, e.g., the rule "append the character 1 to the end" is converted to "delete the last character". When some rules (e.g., "delete the last character") cannot be directly inverse due to the unknown character for "append", we use a wildcard placeholder (e.g., "*") in the inferred words and predict the characters behind wildcard placeholder positions based on the probability distributions. Based on our empirical evaluation, we can achieve at least 51.1% inferred accuracy of based words (see Appendix E for details).

Deployment. We deploy the CPG and TPG models in the

[8] https://www.fox29.com/news/the-20-most-common-passwords-leaked-from-data-breaches-did-yours-make-the-list.

client-side, and the reversed ARPG model in the backend based on their varying usage. For CPG and TPG, we convert the models to browser-usable json-format model via tensorflow.js, whose model size is 9.4 MB and 28 MB, respectively. We count that the average inference time is 31.2 ms and 109.95 ms per password for the respective client-side CPG and TPG models. We count the average inference time is 1.25 ms per password in the reversed ARPG model on backend. We only deploy the reversed ARPG in the server side with python-based interface, since the reversed ARPG should be combined with a password leakage checkup [23, 47].

As shown in Figure 5, HPSM can serve as defense measures, where users can improve their passwords by changing the labeled weak characters (red) with the highest security gains, check against TPG attacks and ARPG attacks (in server-side).

7 Discussion

Password bi-directionality. Bi-directional transformers work better in extracting text features than leading approaches like autoencoders (*CWAE*), RNNs (*Pass2path*) or CNNs (*ADaMs*), which are typically used in real-world guessing attacks. We reasonably conclude that a big part of the success goes from the ability of capturing bi-directional passwords based on the self-attention mechanism. The prior work [22] has already set uni-directional transformers, and showed the superiority of the bi-directional training mechanisms in general guessing tasks. As visualized in Figure 2, passwords exhibit the bi-directionality that weights characters differently.

Besides, pre-training can play an important role in improving the guessing efficiencies. A deep learning model is generally composed of general pre-trained layers and task-specific layers, where the contextualized embedding layers from the pre-trained layers can provide more contextual information for task-relevant layers.

Suitable guessing scenarios of pre-training. The pre-trained models (trained on either natural language or passwords) can easily improve the guessing performance in untargeted guessing scenarios (e.g., CPG or ARPG) with the aid of priori knowledge. While both pre-trained models can only provide marginal gains for targeted attacks (e.g., TPG). We can directly use *PassBERT for TPG. Targeted guessing scenarios always have their own highly task-relevant objective that learns less informative feature from the general pre-trained parameters. Untargeted guessing scenarios are usually more relevant with general password features that are implicitly encoded in the pre-trained parameters. Further, pre-training can play a much larger role when the relevant supervision signals are hard to obtain (*super-rare* pivots in CPG).

Effect of the pre-trained natural language and password models. Compared with the pre-trained natural language model (Vanilla BERT), pre-trained password model (PassBERT) generally plays a larger role in untargeted guessing scenarios, and a similar role in targeted attacks, showcasing the effectiveness of pre-training on password-specific corpus. The pre-trained natural language model is also beneficial to partial password guessing attacks. This is because that passwords can serve as a specific natural language, as the prior study [53] also showed that both natural language and passwords follow the Zipf distribution. Besides, the recent work [60] explored the semantic difference between passwords and natural language.

Practical takeaways. This paper offers the following takeaways: (1) We demonstrate the potential threat from real-world guessing attacks (e.g., CPG, TPG and ARPG), which can significantly threaten password-based authentications. (2) Bi-directional transformers can significantly improve real-world password attacks, with the potential of generalizing more guessing attacks. (3) We explain that pre-training on an unsupervised task (e.g., MLM), either upon the web corpus or the passwords, are generally beneficial to other guessing attacks in the password domain. We show the roles played by pre-training in the password domain, although pre-training has been widely shown effective in NLP. (4) The advanced attacks lead to valuable ideas in the design of PSMs, and push PSM towards comprehensive strength evaluation like *HPSM*.

Future work. In future, we aim to explore more potential of the natural language techniques in a broad range of guessing attacks (e.g., general attacks and more real-world guessing attacks). Particularly, we also consider i) combine Vanilla BERT and PassBERT in guessing attacks, ii) explore a larger pre-trained guessing framework. We believe that this line of exploration would be promising in the password domain.

8 Conclusions

In this paper, we propose a bi-directional-transformer-based password guessing framework, referred to as PassBERT, to improve real-world guessing attacks. We provide an off-the-shelf pre-trained password model, and design task-specific fine-tuning approaches to tailor the pre-trained password model to three specific models of CPG, TPG and ARPG. The experimental results show that the three fine-tuned models can outperform previous best-reported models by 14.53%, 21.82% and 4.86%, respectively, demonstrating the effectiveness of bi-directional transformers on real-world guessing attacks. Finally, we propose a *hybrid password strength meter* with sub-second latency to mitigate the risks from the three real-world guessing attacks.

Acknowledgement

We thank Mr. Luwei Cheng, the anonymous shepherd and all anonymous reviewers for their insightful comments. This paper is supported by NSFC (Grant NO.: U1836207, 62172100) and STCSM Key Projects (Grant NO.: 21DZ1201400, 21511101600). The corresponding author is Weili Han.

References

[1] Neopets. https://www.neopets.com/, 2011.

[2] 000webhost. https://www.000webhost.com/, 2015.

[3] Anatomy of a hack: How crackers ransack passwords like "qeadzcwrsfxv1331", 2013. https://tinyurl.com/y9jw4va6.

[4] J. Blocki, B. Harsha, and S. Zhou. On the economics of offline password cracking. In *Proceedings of 2018 IEEE Symposium on Security and Privacy (SP 2018), San Francisco, California, USA*, pages 853–871. IEEE Computer Society, 2018.

[5] J. Bonneau, C. Herley, P. C. v. Oorschot, and F. Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Proceedings of 2012 IEEE Symposium on Security and Privacy (SP 2012)*, pages 553–567, 2012.

[6] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. Passwords and the evolution of imperfect authentication. *Commun. ACM*, 58(7):78–87, 2015.

[7] J. Casal. 1.4 billion cleartext credentials discovered in a single database., 2017.

[8] Cit0day data breach. https://www.bitdefender.com/, 2020.

[9] Collection#1 data breach. https://www.troyhunt.com/the-773-million-record-collection-1-data-reach/, 2019.

[10] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang. The tangled web of password reuse. In *Proceedings of NDSS*, 2014.

[11] M. Dell'Amico and M. Filippone. Monte carlo strength evaluation: Fast and reliable password checking. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15), Denver, CO, USA, October 12-16, 2015*, pages 158–169.

[12] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019), Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186.

[13] M. Ding, M. Chen, W. Chen, and L. Cai. English cloze test based on BERT. In *Proceedings of 14th International Conference of Knowledge Science, Engineering and Management (KSEM 2021), Part II, Tokyo, Japan, August 14-16, 2021*, volume 12816 of *Lecture Notes in Computer Science*, pages 41–51.

[14] L. Dong, N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou, and H. Hon. Unified language model pre-training for natural language understanding and generation. In *Proceedings of Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems (NeurIPS 2019), December 8-14, 2019, Vancouver, BC, Canada*, pages 13042–13054.

[15] W. Han, Z. Li, M. Ni, G. Gu, and W. Xu. Shadow attacks based on password reuses: A quantitative empirical analysis. *IEEE Trans. Dependable Sec. Comput.*, 15(2):309–320, 2018.

[16] W. Han, M. Xu, J. Zhang, C. Wang, K. Zhang, and X. S. Wang. Transpcfg: Transferring the grammars from short passwords to guess long passwords effectively. *IEEE Trans. Inf. Forensics Secur.*, 16:451–465, 2021.

[17] Jens, Steube Hashcat. https://hashcat.net/hashcat/, 2009-., 2009-.

[18] The details of the permutation functions in hashcat. https://hashcat.net/wiki/doku.php?id=rule_based_attack, 2022.

[19] Y. He, E. E. Alem, and W. Wang. Hybritus: a password strength checker by ensemble learning from the query feedbacks of websites. *Frontiers Comput. Sci.*, 14(3):143802, 2020.

[20] B. Hitaj, P. Gasti, G. Ateniese, and F. Pérez-Cruz. Passgan: A deep learning approach for password guessing. In *Proceedings of 17th International Conference of Applied Cryptography and Network Security (ACNS 2019), Bogota, Colombia, June 5-7, 2019*, volume 11464 of *Lecture Notes in Computer Science*, pages 217–237.

[21] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, and J. López. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *Proceedings of IEEE Symposium on Security and Privacy (SP 2012), 21-23 May 2012, San Francisco, California, USA*, pages 523–537.

[22] H. Li, M. Chen, S. Yan, C. Jia, and Z. Li. Password guessing via neural language modeling. In *Proceedings of Machine Learning for Cyber Security - Second International Conference, (ML4CS 2019), Xi'an, China, September 19-21, 2019*, volume 11806 of *Lecture Notes in Computer Science*, pages 78–93.

[23] L. Li, B. Pal, J. Ali, N. Sullivan, R. Chatterjee, and T. Ristenpart. Protocols for checking compromised credentials. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS'19), London, UK, November 11-15, 2019*, pages 1387–1403.

[24] Y. Li, H. Wang, and K. Sun. A study of personal information in human-chosen passwords and its security implications. In *Proceedings of 35th Annual IEEE International Conference on Computer Communications (INFOCOM 2016), San Francisco, CA, USA, April 10-14, 2016*, pages 1–9.

[25] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proceedings of IEEE International Conference on Computer Vision (ICCV 2017), Venice, Italy, October 22-29, 2017*, pages 2999–3007.

[26] E. Liu, A. Nakanishi, M. Golla, D. Cash, and B. Ur. Reasoning analytically about password-cracking software. In *Proceedings of 2019 IEEE Symposium on Security and Privacy (SP 2019), San Francisco, CA, USA, May 19-23, 2019*, pages 380–397.

[27] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[28] J. Ma, W. Yang, M. Luo, and N. Li. A study of probabilistic password models. In *2014 IEEE Symposium on Security and Privacy (SP 2014), Berkeley, CA, USA, May 18-21, 2014*, pages 689–704.

[29] P. Marquardt, A. Verma, H. Carter, and P. Traynor. (sp)iphone: decoding vibrations from nearby keyboards using mobile phone accelerometers. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS'11), Chicago, Illinois, USA, October 17-21, 2011*, pages 551–562.

[30] W. Melicher, B. Ur, S. Komanduri, L. Bauer, N. Christin, and L. F. Cranor. Fast, lean, and accurate: Modeling password guessability using neural networks. In *Proceedings of 2017 USENIX Annual Technical Conference (USENIX ATC 2017), Santa Clara, CA, USA, July 12-14, 2017*.

[31] R. Morris and K. Thompson. Password security: A case history. *Communications of the ACM*, 22(11):594–597, 1979.

[32] A. Narayanan and V. Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05)*, pages 364–372, New York, NY, USA, 2005. ACM.

[33] P. Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 617–630. Springer, 2003.

[34] K. Omelianchuk, V. Atrasevych, A. N. Chernodub, and O. Skurzhanskyi. Gector - grammatical error correction: Tag, not rewrite. In *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications, BEA@ACL 2020, Online, July 10, 2020*, pages 163–170.

[35] B. Pal, T. Daniel, R. Chatterjee, and T. Ristenpart. Beyond credential stuffing: Password similarity models using neural networks. In *Proceedings of 2019 IEEE Symposium on Security and Privacy (SP 2019), San Francisco, CA, USA, May 19-23, 2019*, pages 417–434.

[36] D. Pasquini, G. Ateniese, and M. Bernaschi. Interpretable probabilistic password strength meters via deep learning. In *Proceedings of 25th European Symposium on Research in Computer Security (ESORICS 2020), Part I, Guildford, UK, September 14-18, 2020*, volume 12308 of *Lecture Notes in Computer Science*, pages 502–522.

[37] D. Pasquini, M. Cianfriglia, G. Ateniese, and M. Bernaschi. Reducing bias in modeling real-world password strength via deep learning and dynamic dictionaries. In *Proceedings of 30th USENIX Security Symposium (USENIX Security 2021), August 11-13, 2021*, pages 821–838.

[38] D. Pasquini, A. Gangwal, G. Ateniese, M. Bernaschi, and M. Conti. Improving password guessing via representation learning. In *Proceedings of 42nd IEEE Symposium on Security and Privacy (SP 2021), San Francisco, CA, USA, 24-27 May 2021*, pages 1382–1399.

[39] D. Pasquini, A. Gangwal, G. Ateniese, M. Bernaschi, and M. Conti. Improving password guessing via representation learning. *IACR Cryptol. ePrint Arch.*, 2019:1188, 2019.

[40] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf, 2018.

[41] Rockyou. https://www.rockyou.com/, 2009.

[42] Rockyou2021. https://www.securedyou.com/rockyou-txt-rockyou2021-download/. https://www.securedyou.com/rockyou-txt-rockyou2021-download/, 2021.

[43] S. Security. 2017 credential spill report. http://info.shapesecurity.com/rs/935-ZAM-778/images/Shape-2017-Credential-Spill-Report.pdf/, 2017.

[44] Y. Shen, X. Ma, Z. Tan, S. Zhang, W. Wang, and W. Lu. Locate and label: A two-stage identifier for nested named entity recognition. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (ACL/IJCNLP 2021), (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 2782–2794.

[45] What you need to know about targeted online guessing. https://www.itproportal.com/features/what-you-need-to-know-about-targeted-online-guessing/, 2017.

[46] D. R. Thomas, S. Pastrana, A. Hutchings, R. Clayton, and A. R. Beresford. Ethical issues in research using datasets of illicit origin. In *Proceedings of the 2017 Internet Measurement Conference (IMC 2017), London, United Kingdom, November 1-3, 2017*, pages 445–462.

[47] K. Thomas, J. Pullman, K. Yeo, A. Raghunathan, P. G. Kelley, L. Invernizzi, B. Benko, T. Pietraszek, S. Patel, D. Boneh, and E. Bursztein. Protecting accounts from credential stuffing with password breach alerting. In *28th USENIX Security Symposium (USENIX Security 2019), Santa Clara, CA, USA, August 14-16, 2019*, pages 1556–1571.

[48] I. O. Tolstikhin, O. Bousquet, S. Gelly, and B. Schölkopf. Wasserstein auto-encoders. *CoRR*, abs/1711.01558, 2017.

[49] B. Ur, J. Bees, S. M. Segreti, L. Bauer, N. Christin, and L. F. Cranor. Do users' perceptions of password security match reality? In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI 2016), San Jose, CA, USA, May 7-12, 2016*, pages 3748–3760.

[50] B. Ur, F. Noma, J. Bees, S. M. Segreti, R. Shay, L. Bauer, N. Christin, and L. F. Cranor. "i added '!' at the end to make it secure": Observing password creation in the lab. In *Proceedings of Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, pages 123–140, Ottawa, 2015.

[51] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems (NerIPS 2017), December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

[52] R. Veras, C. Collins, and J. Thorpe. On semantic patterns of passwords and their security impact. In *21st Annual Network and Distributed System Security Symposium (NDSS 2014), San Diego, California, USA, February 23-26, 2014*.

[53] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian. Zipf's law in passwords. *IEEE Trans. Information Forensics and Security*, 12(11):2776–2791, 2017.

[54] D. Wang, P. Wang, D. He, and Y. Tian. Birthday, name and bifacial-security: Understanding passwords of chinese web users. In *Proceedings of 28th USENIX Security Symposium (USENIX Security 2019), Santa Clara, CA, USA, August 14-16, 2019*, pages 1537–1555.

[55] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang. Targeted online password guessing: An underestimated threat. In *Proceedings of 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16), Vienna, Austria, October 24-28, 2016*, pages 1242–1254.

[56] M. Weir. The version of 4.1 for pcfg models, 2019. https://github.com/lakiw/pcfg_cracker.

[57] M. Weir, S. Aggarwal, B. d. Medeiros, and B. Glodek. Password cracking using probabilistic context-free grammars. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy (SP 2019)*, pages 391–405, Washington, DC, USA.

[58] D. L. Wheeler. zxcvbn: Low-budget password strength estimation. In *Proceedings of 25th USENIX Security Symposium (USENIX Security 16)*, pages 157–173, Austin, TX, 2016.

[59] A. Williams, N. Nangia, and S. R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2018), New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1*, pages 1112–1122.

[60] M. Xu, C. Wang, J. Yu, J. Zhang, K. Zhang, and W. Han. Chunk-level password guessing: Towards modeling refined password composition representations. In *Proceedings of 2021 ACM SIGSAC Conference on Computer*

and Communications Security (CCS'21), Virtual Event, Republic of Korea, November 15 - 19, 2021, pages 5–20.

[61] J. J. Yan, A. F. Blackwell, R. J. Anderson, and A. Grant. Password memorability and security: empirical results. *IEEE Security Privacy*, 2(5):25–31, 2004.

[62] H. Zhang, C. Wang, W. Ruan, J. Zhang, M. Xu, and W. Han. Digit semantics based optimization for practical password cracking tools. In *ACSAC '21: Annual Computer Security Applications Conference, Virtual Event, USA, December 6 - 10, 2021*, pages 513–527. ACM, 2021.

[63] Y. Zhang, Y. Mao, M. Xu, F. Xu, and S. Zhong. Towards thwarting template side-channel attacks in secure cloud deduplications. *IEEE Trans. Dependable Secur. Comput.*, 18(3):1008–1018, 2021.

A Details of Input Embeddings

The character embedding layer will convert each character into a multi-dimensional (e.g., 256) vector representation via its lookup-table, e.g., a (99×256) lookup vector expression. Similarly, the position embedding layer converts the order into a multi-dimensional vector, which learns the sequence's order information via its lookup-table of a (34×256) vector expression. Finally, we sum up its character and position embedding as the final input embedding. which results in that the same characters in different positions can have different embeddings due to position embeddings.

B Supplemented Model Architecture Details

We show the architecture of our pre-trained model in Table 9, where includes the pre-trained layers and task-specific layers with the output shape of each layer. The size of our pre-trained password model's parameters is 2,332,259. In the output shape column, *batch-size* refers to the number of passwords processed per iteration and *seq-length* denotes the maximum number of characters in a sequence. Since we consider the longest passwords to 32 characters, the *seq-length* is thus 34 with the starting and ending symbols. Consistent with BERT [27, 51], we only adopt the encoder mechanism to learn general password features by masking mechanism. Since passwords are more short than natural language, we use 4 transformer blocks with dimensions of 256.

The CPG model architecture is shown in Table 10, which is the same with the pre-trained model. We show the TPG and ARPG architecture in Table 11 and 12, respectively. The TPG model produces edit operations in *character-level*, i.e., each character produces a operation among 9,122 edit operations given a password. The ARPG model predicts the adaptive rules in *password-level*, i.e., the model selects adaptive rules given a password. Moreover, we list the hyper-parameters that we used in training the pre-trained model, CPG model, TPG model and ARPG model in Table 13.

Table 9: Architecture of the pre-trained password model, including pre-trained layers and task-specific layers. Pre-trained layers consist of the input and embedding layers and all transformer blocks. Task-specific layers consist of a full connected layer and the output layer.

Transformer encoder (2,332,259)	
Layers	output shape
`Input layer`	[batch-size, seq-length]
`Embedding layer`	[batch-size, seq-length, 256]
`Transformer block`	[batch-size, seq-length, 256]
`Transformer block`	[batch-size, seq-length, 256]
`Transformer block`	[batch-size, seq-length, 256]
`Transformer block`	[batch-size, seq-length, 256]
`FullyConnected`	[batch-size, seq-length, 256]
`Output layer`	[batch-size, seq-length, 99]

Table 10: Architecture of the CPG attack model.

CPG model (2,332,259)	
Layers	output shape
`Pre-trained layers`	[batch-size, seq-length, 256]
`FullyConnected`	[batch-size, seq-length, 256]
`Output layer`	[batch-size, seq-length, 99]

C Computational Performance in ARPG

We calculate the number of rules' score per second calculated by PassBERT and *ADaMs* in the same machine. We make these experiments three times with the randomly sampled 10^7 words in `Rockyou-2009` to guarantee the stability of testing time. We show the results in Table 14, from which we can observe that the two models have similar inference speed in *PasswordPro*, while our models are slightly faster in *Generated*. This is because our ARPG models are lightweight, i.e., the network size is 3,998,000 (*PasswordPro*) and 9,952,904 (*Generated*), compared with *ADaMs* of 12,625,920 (*PasswordPro*) and 23,026,688 (*Generated)*. Although transformers tend to be slower than CNNs due to the quadratic time/memory cost of self-attention in general case, resulting the sound latency. Given that our ARPG models are just slightly faster than *ADaMs*, and can improve the guessing effi-

Table 11: Architecture of the TPG attack model.

TPG model (7,077,026)	
Layers	output shape
`Pre-trained layers`	[batch-size, seq-length, 256]
`FullyConnected`	[batch-size, seq-length, 256]
`Output layer`	[batch-size, seq-length, 9122]

Table 12: Architecture of the ARPG attack model. We train two networks for two rules-set of *PasswordPro* and *Generated*, whose size of neural networks is 3,998,000 and 9,952,904.

ARPG model (3,998,000; 9,952,904)	
Layers	output shape
`Pre-trained layers`	[batch-size, seq-length, 256]
`FullyConnected`	[batch-size, seq-length, 256]
`Output Layer`	[batch-size, rules-set-size]

Table 13: Hyper-parameters used in training our models.

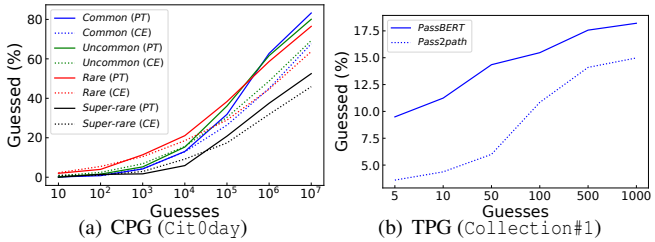
Hyperparameter	Value			
	Pre-trained model	CPG	TPG	ARPG
Attention heads	2	2	2	2
Learning rate	0.002	2×10^{-5}	10^{-5}	10^{-5}
Optimizer	Adam	Adam	Adam	Adam
Dropout	0.1	0.1	0.1	0.1
Seq-length	32	32	32	32
batch-size	256	512	128	256
Epoch	2,625	125	3	4
batch-size	256	256	256	256
seq-length	34	34	34	34

ciencies, PassBERT can become a better model in real-world attack practice. We also supplement the implementation of our ARPG attacks in the following part.

Implementation of ARPG. We modify the code to scan all dictionary words with the selected rules being applied. We use the *batch processing technique* to improve throughout, i.e., we give read a batch of words from the dictionary and take them as input to our neural network, and then, for each word in the batch, we apply only the rules whose values are greater than the threshold ($\mathcal{V}$). In the dynamic version, we add the cracked passwords on the tail of the dictionary with the same batch strategies. Same as *ADaMs*, we use the batch-size of 4,096 dictionary words.

D Evaluation of Smaller Guesses in CPG and TPG

We show the guessing performance under comprehensive guesses in Figure 6 for CPG and TPG, from which we can observe that CPG performs better when guesses become larger, while TPG achieves significant improvement when guesses are smaller. We believe that current evaluation metrics (same as *CWAE* and *Pass2path*) can be more appropriate since CPG and TPG generally fall into different guessing scenarios, i.e., CPG cares about cracking more associated passwords that satisfy the pivot (perhaps obtained from malicious monitoring in a surveillance camera), while TPG focuses on targeted online guessing scenarios limited to smaller guesses. We observe that CPG cracks less passwords in smaller guesses (e.g., around 10% cracking rates in 1,000 guesses), we argue that it can be more suitable to compare the efficiency improvements when both models can reach their higher cracking rates (e.g., around 50% cracking rates).



(a) CPG (`Cit0day`) (b) TPG (`Collection#1`)

Figure 6: Evaluation of more guesses in CPG and TPG. PT refers to PassBERT, and CE denotes *CWAE*.

Table 14: Number of rules for a base word computed per second (r/s) for different networks. The values are obtained under the same computation power.

PassBERT		*ADaMs*	
PasswordPro	*Generated*	*PasswordPro*	*Generated*
15.4 million r/s	55.5 million r/s	15.5 million r/s	44.1 million r/s

E Accuracy of Reversed ARPG in HPSM

We evaluate the inferred accuracy of base words given an input password. We sample n words in the dictionary of `Rockyou-2009` to crack the target dataset of `Neopets`, then we collect all the cracked passwords as a set T. For each password in T, we infer its possible top k words, and compared with n wordlists to finally calculate the inferred accuracy by $\frac{|T|*k}{n}$ (the n and k are 10,000 and 100 in our experiments, respectively). The results show that we can achieve an inferred accuracy of 51.1%, the maximum theoretical inferred accuracy of 75.9% where we assume that all rules can be inversed correctly. When we assume that all these inferred wordlists with wildcard placeholders can be correctly recovered (possibly via the CPG models), we can achieve the inferring accuracy of 67.3%.