applied sciences

MDPI

Article

Generating Optimized Guessing Candidates toward Better Password Cracking from Multi-Dictionaries Using Relativistic GAN

Sungyup Nam, Seungho Jeon and Jongsub Moon *

Graduate School of Information Security, Korea University, Seoul 02841, Korea; synam@korea.ac.kr (S.N.); ohgnu90@korea.ac.kr (S.J.)

* Correspondence: jsmoon@korea.ac.kr

Received: 8 September 2020; Accepted: 13 October 2020; Published: 19 October 2020

Abstract: Despite their well-known weaknesses, passwords are still the de-facto authentication method for most online systems. Due to its importance, password cracking has been vibrantly researched both for offensive and defensive purposes. Hashcat and John the Ripper are the most popular cracking tools, allowing users to crack millions of passwords in a short time. However, their rule-based cracking has an explicit limitation of depending on password-cracking experts to come up with creative rules. To overcome this limitation, a recent trend has been to apply machine learning techniques to research on password cracking. For instance, state-of-the-art password guessing studies such as PassGAN and rPassGAN adopted a Generative Adversarial Network (GAN) and used it to generate high-quality password guesses without knowledge of password structures. However, compared with the probabilistic context-free grammar (PCFG), rPassGAN shows inferior password cracking performance in some cases. It was also observed that each password cracker has its own cracking space that does not overlap with other models. This observation led us to realize that an optimized candidate dictionary can be made by combining the password candidates generated by multiple password generation models. In this paper, we suggest a deep learning-based approach called REDPACK that addresses the weakness of the cutting-edge cracking tools based on GAN. To this end, REDPACK combines multiple password candidate generator models in an effective way. Our approach uses the discriminator of rPassGAN as the password selector. Then, by collecting passwords selectively, our model achieves a more realistic password candidate dictionary. Also, REDPACK improves password cracking performance by incorporating both the generator and the discriminator of GAN. We evaluated our system on various datasets with password candidates composed of symbols, digits, upper and lowercase letters. The results clearly show that our approach outperforms all existing approaches, including rule-based Hashcat, GAN-based PassGAN, and probability-based PCFG. The proposed model was also able to reduce the number of password candidates by up to 65%, with only 20% cracking performance loss compared to the union set of passwords cracked by multiple-generation models.

Keywords: password cracking; relativistic discriminator; rPassGAN

1. Introduction

The password is the de-facto authentication method. It is popular due to its simplicity to implement and easiness to use. Password authentication ultimately depends on human memory. Thus, as revealed by the leaked passwords of websites, such as Rockyou, people tend to generate easy-to-remember passwords, primarily composed of common English words or names [1–3]. Password cracking utilities provide many functions for attacking weak passwords when password hashes are available. Cracking software's effectiveness depends on being able to hash a large number of password candidates and compare the hashed value with target hashes. Notably, a heterogeneous computing system using multiple GPUs and GPU libraries (CUDA, OpenCL) accelerates the hash computing speed. To maximize password cracking effectiveness, instead of trying all the possible character combinations (exhaustive attack or brute-force attack), password cracking tools use words that users are likely to generate, as can be inferred from cracked hashes dictionaries and plaintext password leaks, as candidate passwords. Such an attacking method is referred to as the dictionary attack. The most commonly used open-sourced password cracking software, such as John the Ripper (JtR) [4] and Hashcat [5], have more advanced options because they supply heuristics character changing grammars, which include addition and deletion of characters, letter transposition, combination of multiple words, mixing of letter case, and leetspeak, for password transformations (e.g., p@ssw0rd). All 31 transformation grammars are defined [6]. These heuristics are stored in several files called rule files. The rule files enable JtR and Hashcat to generate several password candidates that people are highly likely to use in the real world.

Although these rule-based heuristic approaches are successful to an extent in practice, they are based on experts' intuitions on how people build their passwords; further, these methods are not based on a systematical analysis of a large number of passwords. Therefore, each technique covers only a limited area of the total password space because it is based on the expert's personal experience and intuition. Further, creating new rules requires specialized expertise; as such, the new rules' quality and their cracking effectiveness depend on the specialist. It is also necessary to manually update these heuristics whenever new password patterns appear. Therefore, these rule-based approaches are not scalable. Furthermore, to crack high-complexity passwords, password cracking methods must first overcome the inherent weakness in rule-based attacks.

Advanced password cracking technology is useful from a security perspective. A state-of-the-art password strength estimator like Zxcvbn [7] has used a password cracking module to evaluate passwords that users input and warn them of easily guessed passwords. This password cracking module prevents a user from creating an easy password to crack. As password cracking tools' performance is getting better, the quality of the password that users need to create becomes higher. Additionally, from an offensive perspective, advanced password cracking methods can be utilized in various ways. On an individual level, people sometimes forgot their passwords. At that time, people need to crack their own passwords. On a societal level, law enforcement agencies need to crack passwords to gather evidence that criminals have encrypted [8].

1.1. Our Previous Approach: Recurrent PassGAN

In our previous paper, we proposed some ways to improve PassGAN [9], which is a deep learning-based password cracking model that is designed to make up for the limited password cracking space of both rule-based approaches and data-driven probabilistic models, such as Markov models e.g., the Ordered Markov ENumerator (OMEN) [2]. The PassGAN, proposed by Hitaj et al. [9], trained deep neural networks to determine the letter distribution of passwords autonomously. Next, PassGAN applied this learned knowledge to generate password candidates that followed the distribution of real passwords. Basically, PassGAN exploits two properties of deep learning. First, deep neural networks are sufficiently expressive to sketch the various letter patterns and context structures used in most user-chosen passwords. Furthermore, they can be trained from data only, unlike legacy machine learning. Therefore, a deep learning model does not require any prior knowledge (we usually refer to this knowledge as a feature) of the passwords' properties and structures. However, a deep learning model can learn features only from the training data. These properties distinguish deep neural networks from other contemporary methods such as Markov and rule-based models. In the case of Markov models, such as OMEN [2], it is assumed that all the relevant characteristics of passwords can be expressed in terms of the n-gram.

However, only password candidates derived from the available rules, which reflect the expert's knowledge and experience, can be guessed for using the rule-based approaches. Hitaj et al. used a GAN [10] as a deep learning model for password guessing [9]. Among the various GAN models, the Wasserstein GAN-gradient penalty (WGAN-GP) [11] model was used in PassGAN. The base deep neural network of the WGAN-GP was a Convolutional Neural Network (CNN)-based residual network. The Recurrent PassGAN (rPassGAN), which was developed in our previous approach, improved the password cracking performance by modifying PassGAN's base neural network type and structure. Our previous study proved that the password cracking performance of rPassGAN [8,12] was better than that of PassGAN. The rPassGAN uses a Recurrent Neural Network (RNN) as its primary deep neural network.

Furthermore, rPassD2CGAN, a dual discriminator version of rPassGAN, outperformed rPassGAN. However, during the training of rPassD2CGAN, it sometimes becomes unstable. Therefore, we overcome this weakness by using another model, rPassD2SGAN. We demonstrated the effectiveness of the password candidates generated by the rPassGAN for enhancing the password strength estimator through several experiments.

1.2. Our New Approach: REDPACK

In our previous studies, the rPassGAN series outperformed PassGAN. Nevertheless, compared with Probabilistic Context-Free Grammar (PCFG) and other Markov models, such as the OMEN [1], rPassGAN sometimes cracked fewer passwords. In this study, we propose a model to maximize the password cracking performance using a small number of guessing. With our model, we can selectively collect more realistic password candidates to improve the efficiency of password cracking. Then, we can make an effective cracking dictionary with these collected candidates. If password candidate dictionaries from various models are available, prior knowledge of the password's properties and structure is not required for selecting a more realistic password candidate. The relativistic average standard GAN [13]'s discriminator operates as an estimator, which evaluates how realistic input passwords are from the various pre-generated candidates. Although a GAN generator is typically used, our model introduces a novel way of using the GAN's discriminator in password guessing. We refer to this model as a relativistic discriminator for password candidates effective pack(REDPACK). We demonstrate that our model can compete favorably with the Hashcat transformation rules (rockyou-30000, dive), PCFG [14], and OMEN [2]. Our new approach's contributions are as follows:

- GAN is a generative deep learning model. Generally, the generator of a GAN is used for producing fake samples. However, the trained discriminator of the GAN is not utilized for the inference. The discriminator is only used in the training phase for training the generator. Our first contribution is that we propose how to utilize the discriminator of the GAN not for training the generator but for enhancing password cracking performance. To the best of our knowledge, our approach is the first model that has been proposed in the field of password cracking. The discriminator of the GAN estimates how realistic password candidates are. Then, we use these estimates as criteria to select the best candidates among the multiple dictionaries.
- If there are multiple pre-generated password candidate dictionaries, REDPACK enables us to make a more effective and efficient password candidate dictionary without any background knowledge about the pre-generated dictionaries. If we provide this efficient and effective password candidate dictionary to a password strength estimator like Zxcvbn, we are able to make the criteria of the password strength estimator stricter [7,8,12].
- Our last contribution is the building of a custom ruleset for REDPACK. This custom ruleset helps the password candidates from REDPACK maximize its cracking performance.

1.3. Organization

The rest of this paper is organized as follows. In Section 2, we provide an overview of the relevant password guessing models. In Section 3, we provide a brief overview of GANs and relativistic average GANs as background knowledge for REDPACK. In Section 4, we explain the concept of REDPACK and describe its architecture. In Section 5, we explain the process of training the hyperparameter configuration. Then, we evaluate the password cracking performance of REDPACK. In this section, we present the results of REDPACK and compare them with other advanced password guessing techniques. Finally, the conclusions are presented in Section 6.

2. Related Works

In this section, we discuss related works regarding password guessing that use categories, probability-based approaches, and deep learning-based approaches. The probability-based approaches are divided into whole-string methods and template-based methods. These terms were defined in Ma et al. [15]

2.1. Rule-based Approaches

Basically, in the password-guessing attack, the adversary attempts to match one or more users' passwords by repeatedly testing large numbers of password candidates. This attack could be conducted in offline or online modes. Password-guessing attacks might be as old as password authentication themselves [16]. JtR [4] and Hashcat [5] are two popular modern password guessing (or cracking) open-source software. This software provides multiple types of password cracking strategies, including exhaustive brute-force attacks, dictionary-based attacks, rule-based attacks (also called hybrid-attack). The rule-based attack generated password candidates by transforming the words in the dictionary according to its own grammar [6,17]; and Markov-model-based attacks [18,19], in which each letter of a password candidate is chosen via a certain statistical and probabilistic process that considers one or more preceding letters, and this model is trained on dictionaries of plaintext passwords. In the practical password cracking field, JtR and Hashcat are promising and useful. Heterogeneous computing technology enhances the cracking performance of both tools. There have been several instances in which well over 90% (370 of 1321 sites' password hash files of Hashes.org have been recovered 90% over in April 2020) of the passwords leaked from online services have been successfully recovered [20]. However, both Hashcat and JtR have an explicit limitation. If their cracking target password is in a complex form and their pre-defined rulesets do not cover the target password, both tools are totally unable to succeed in cracking the target password.

2.2. Markov-Based Approaches & PCFG

In the stochastic approach, a method incorporating the Markov model has been proposed. Narayanan et al. proposed a method of generating a fake password (or password candidates) using the Markov model for the alphanumeric characters and the Turing machine for special symbols [21]. This method's the fundamental idea is that frequently used words and patterns are limited to easy-to-remember passwords; further, the passwords in this space are included in a specific probability distribution of alphanumeric combinations. Narayanan et al. used the Markov model as a filter to eliminate the low-probability password candidates. Furthermore, they applied the rainbow table concept (time-space tradeoff) to speed up password cracking. Finally, to handle the passwords with special symbols, the Turing machine concept was adopted. Based on a password generation rule, various alphabets and numbers were combined into regular expressions, and the probability for each combination created was defined. This pioneering work was subsequently extended by Ma et al. [15] and Dürmuth et al. [2].

Dürmuth et al. [2] proposed an efficient password guesser based on Narayanan's model. This model was called the OMEN. Basically, OMEN aimed to improve the cracking speed of

Narayanan's model. It incorporated an improved enumeration algorithm called "enumPwd", which enabled it to produce candidates in order of probability by implementing multiple bins. In each bin, candidates of similar probabilities were stored.

The most important aspect of these password guessing studies was incorporating the PCFG concept into the password-guessing method. PCFG was invented by Weir et al. [22] originally as a password guesser. Current complex passwords have grammatical structure. This grammatical structure is a combination of alphanumerical sequences, special characters, and keyboard-walks. PCFG analyzes the grammatical structure of leaked passwords and calculates the distribution probability from these leaked passwords. For generating password candidates, PCFG uses the grammatical structure in order of the probability. Recently, several studies have improved upon the performance of the PCFG [23–25]. Based on common current password usage patterns and on government recommendations [26], password guessing must be able to produce grammatical structures that include not only simple alphabetical and numerical combinations but also complex combinations that include special characters and keyboard-walks. PCFG has enabled us to generate these complicated password patterns. In experiments, PCFG exhibited a higher cracking success rate than dictionary-based attacks using Hashcat built-in rules. This method could expand the cracking area of the password space effectively. Houshmand et al. [24] focused on improving the cracking performance for keyboard-walk structures and employed a smoothing technique. In their experiments, they achieved good performance on cracking keyboard-walk patterned passwords. Furthermore, Houshmand et al. proposed the use of PCFG as a target-oriented cracking approach [25].

Finally, Ma et al. [15] attempted to maximize the performance of the probability-based password cracking approaches through optimized configuration and usage. Additionally, they proposed a new way of measuring the password cracking performance. Ma et al. categorized password guessing methods as whole-string or template-based. Narayanan et al.'s model and OMEN were included in the whole-string models. PCFG was classed as a template-based model. Ma et al. attempted to derive each model's best configuration and introduced the n-gram model for statistical language modeling into password modeling. In the experiments conducted by Ma et al., in several cases, the whole-string Markov models outperformed PCFG. However, specific configurations, such as smoothing, enhanced the performance of PCFG in such a way that it outperformed the whole-string approach.

2.3. Deep Learning-Based Approaches

The first password guessing method that employed deep learning was proposed by Melicher et al. [27]. They incorporated an RNN [28] into their model. The purpose of Melicher's study was to enhance the password strength estimator based on a deep-learning password guesser. RNNs, a deep learning neural network that has been popularly adopted in the field of Natural Language Processing (NLP), usually exhibit good performance in various applications, such as chat-bots, translation, and auto-completion. In Melicher et al.'s method [27], leaked passwords were used as the training data. Subsequently, a guessing candidate was produced letter by letter. In the RNN model, the characters that constitute the password are based on all the previously selected characters.

In addition to the method by Melicher et al. [27], Hitaj et al. proposed PassGAN [9] based on the most popular generative model, a GAN. In detail, Hitaj et al.'s PassGAN employed IWGAN [11]. Throughout their experiments, PassGAN competed favorably with state-of-the-art password generation tools. PassGAN is the first approach to apply GAN to the password guessing problem. PassGAN used the original IWGAN model, which used the WGAN-GP cost function, for training the generator of the GAN without any modifications. In the original IWGAN model, both the generator and discriminator of the GAN used a Convolutional Neural Network (CNN) as their primary components. However, CNNs are usually used for processing images in deep learning studies. So, in our previous studies, a Recurrent Neural Network (RNN) based model was used to improve the password cracking performance [8,12]. We named our models rPassGAN and rPassD2SGAN, which indicates a dual discriminator architecture. When Jensen-Shannon Divergence (JSD) is high between the training dataset and the cracking target set, rPassGAN cracked more passwords than PCFG. That is to say, the general performance of rPassGAN is better than PCFG. Otherwise, PCFG cracked more passwords than rPassGAN. Although PassGAN could not outperform other password guessing models, Hitaj et al. emphasized that their model was able to crack some passwords that could not be cracked by other stochastic models [9]. rPassGAN was able to achieve similar results. Furthermore, this trend can be observed among the RNN-based PassGAN models [8,12].

3. Background for REDPACK

In this section, we provide a brief overview of a standard GAN and a relativistic average GAN. First, we describe GAN's development history, following which we explain relativistic GANs, which are at the core of our model.

3.1. Generative Adversarial Networks

GANs have brought about remarkable advances in the field of deep learning. A GAN is composed of two neural networks. The first is a generative deep neural network G, which performs the main task of training. The other is a discriminative deep neural network D, which functions as the supervisor. Given an n-sized input batch $I = \{x_1, x_2, ..., x_n\}$, the main goal of G is to generate fake samples that D can confuse with the real ones based on the responses of D; otherwise, the goal is to learn to distinguish the fake samples from G from the real ones coming from I. The optimization problem for GANs is a minimax problem. Goodfellow et al. [10] solved this problem by allowing the GAN to have a global optimum when the distribution of fake samples produced by G was mathematically identical to the distribution of the given real data. When z is a noise input from the uniform distribution, the minimax problem can be expressed as follows:

$$\underset{G}{min}\,\underset{D}{max}\,V(D,G) = \underset{x\sim\mathbb{P}_{data}(x)}{\mathbb{E}}[logD(x)] + \underset{z\sim\mathbb{P}_z(z)}{\mathbb{E}}[log(1-D(G(z)))] \tag{1}$$

The learning of the generator can be regarded as optimized when D cannot distinguish between the fake samples generated by G and the real samples from I. Since Goodfellow et al. proposed the initiative GAN, various GAN models with more stable training performance have been proposed. Among these GAN models, the PassGAN-related ones are the Wasserstein GAN (WGAN) [29] and the improved Wasserstein GAN (IWGAN) [11]. WGAN, which was introduced by Arjovsky et al., improves the training stability of a standard GAN by employing the Wasserstein distance for loss. The benefits of this approach include reduced mode collapse and meaningful learning curves, which are helpful in identifying optimal hyperparameters. WGAN incorporates a new cost function; however, experiments on the WGAN focus on generating realistic images. Gulrajani et al. [11] proposed IWGAN to find the global optimum more effectively, compared to WGAN. They introduced the concept of the gradient penalty to replace the gradient clipping of WGAN. Gulrajani et al. proposed the use of IWGAN to solve the text generation problem. In Gulrajani et al.'s IWGAN, both G and D consisted of simple residual CNNs. The residual architecture makes the training of the GAN fast and stable [30,31]. G takes as input a latent noise vector, transforms it by forwarding it through its convolutional layers, and outputs a sequence of 32 one-hot character vectors. The output layer of G adopts a softmax nonlinearity function and forwards it to D. Each output character of a fake sample is determined by the result of the *argmax* function, which takes each output vector generated by G as input. The IWGAN experiment motivated Hitaj et al. [9] to apply IWGAN to the password guessing problem. They referred to the model that they created as PassGAN.

3.2. Relativistic average GAN

The applications of GAN, a groundbreaking framework for learning generative models, are varied. However, standard GAN (SGAN), originally proposed by Goodfellow et al. [10], is unstable in the learning phases, and optimizing the model is difficult. Many alternatives, including WGAN, have been proposed to mitigate this problem. GAN-based models can be broadly divided into integral probability metric (IPM)-based GANs and non-IPM-based GANs. Generally, IPM constraints provide stability for training the GAN. Jolicoeur [13] analyzed the loss function of SGAN to analyze the limitations of the non-IPM-based GAN and proposed a relativistic GAN to address this. Goodfellow et al. [10] proved that GAN training attains the global optimum when the discriminator classifies real data with 0.5 probability. However, in many cases, the discriminator classifies both real and fake data as real, which is disadvantageous in training a good generator. This is because the discriminator is not aware that half of the data in the learning process is fake; the IPM-based GAN is relatively stable during learning because it implicitly accounts for this fact. From the perspective of divergence minimization, the discriminator is trained to increase $D(x_f)$, whereas $D(x_r)$ does not decrease accordingly, where x_r and x_f denote the real and fake data, respectively. To address this issue, Jolicoeur [13] designed the output of the discriminator to depend on both the real and fake data:

$$D(x_r, x_f) = \sigma(C(x_r) - C(x_f)) \tag{2}$$

Where $C(x)$ is the presumed critic ($D(x) = \sigma(C(x))$). Equation (2) can be interpreted as the discriminator's estimation of the probability of the given real data being more realistic than the fake data ($D(x_f, x_r)$ can be interpreted in the opposite manner). If the discriminator is set according to Equation (2), unlike the generator of SGAN, which relies solely on the fake data, the generator from the relativistic GAN will depend on both the real and fake data. However, it has $O(m^2)$ complexity when calculating the loss (m means the size of the mini-batch) because it calculates the pair-wise differences of the critic between the real and fake data in the mini-batch. To solve this problem, Jolicoeur [13] proposed a relativistic average GAN (RaGAN), which takes the expectation of the opposite type of data for some given data. The RaGAN uses the following loss functions to learn the discriminator and generator:

$$\begin{aligned} L_D &= -\mathbb{E}_{x_r}[\log(\sigma(C(x_r) - \mathbb{E}_{x_f}[C(x_f)]))] - \mathbb{E}_{x_f}[\log(1 - \sigma(C(x_f) - \mathbb{E}_{x_r}[C(x_r)]))] \\ L_G &= -\mathbb{E}_{x_f}[\log(\sigma(C(x_f) - \mathbb{E}_{x_r}[C(x_r)]))] - \mathbb{E}_{x_r}[\log(1 - \sigma(C(x_r) - \mathbb{E}_{x_f}[C(x_f)]))] \end{aligned} \tag{3}$$

where L_D and L_G represent the loss for learning the discriminator and generator, respectively. Equation (3) has $O(m)$ complexity. The discriminator of RaGAN estimates the probability of some given data being more realistic than the opposite type of data, on average. Using different datasets, Jolicoeur [13] showed that training RaGAN, compared to other GAN models, was faster and more reliable, and the generator of RaGAN generated high-quality fake data.

4. Proposed Model: REDPACK

In this section, we describe the structure of REDPACK. At first, we explain the overview of REDPACK. Next, the discriminator training structure of REDPACK and the password candidate selecting structure of REDPACK will be described in detail.

4.1. Overview

Generally, GAN models are trained for solving generative problems. So, after the training of the GAN model is finished, the generator of the GAN should be used for achieving its goal. In our previous research [8,12], we used the generator of our GAN for producing more realistic password candidates. However, in the case of REDPACK, the discriminator of the GAN is used after finishing the training of the model. Figure 1 shows the train phase and selection phase. In the training phase, the generator produces fake passwords. Then, the discriminator tries to distinguish fake passwords from real passwords. Next, the discriminator sends feedback to the generator. By using the feedback from the discriminator, the generator learns how to make fake passwords that are almost the same as real passwords. After training the generator, the discriminator should solve the more difficult problem of differentiating real passwords from fake passwords than it was in the previous training step. This process makes the discriminator stronger. Then, the training process continues until the training parameters converge. In the selection phase, multiple password generators can be used for producing the password candidates. As a result of the training phase, the discriminator calculates the probability of how realistic each generator's password candidates are. The closer the probability is to one, the closer the password candidates are to realistic passwords. In the final step, the password candidates of the highest probability are supplied to a password cracking tool like Hashcat (or John the Ripper). From an implementation perspective, there is no limit to the number of password candidate generators that can be used in the selection phase. In this paper, we use three or four password candidate generators for REDPACK.

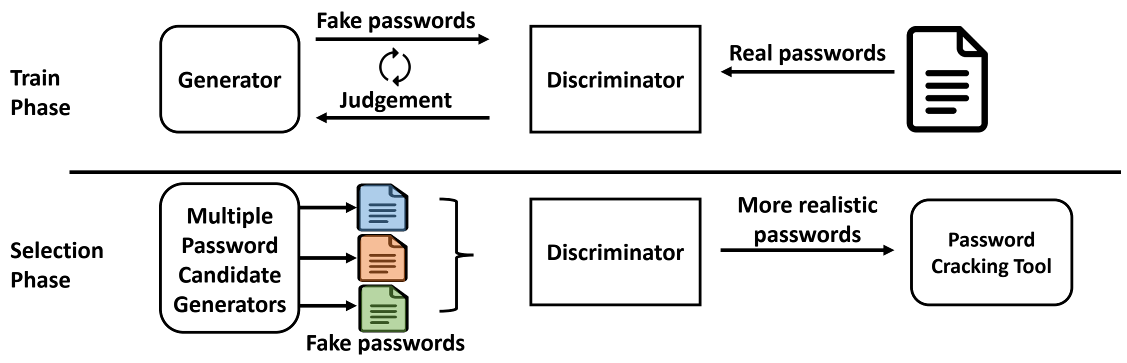


Figure 1. Overview Diagram of REDPACK.

4.2. The Discriminator Training Structure

Figure 2 shows a process to optimize the parameters of an RNN-based GAN. The RNN-based GAN is built upon a generator (G) and a discriminator (D). We employ an RNN as the base model of the GAN because passwords are a type of sequential data. As with a standard GAN [10], G is trained to generate fake passwords, but these are very similar to real passwords; D tries to distinguish real passwords from the fake passwords. However, our RNN-based GAN adopts the concept of both relativistic average GAN [13] and IWGAN [11] to achieve a more powerful discriminator. In Figure 2, G produces fake passwords from a given arbitrary noise distribution $\mathbb{P}_z(z)$, which is depicted by a green path; following Equation (3), D determines the fake passwords as real if and only if the critic for the fake password is larger than the real one described as blue storage in the figure. Likewise, the real passwords on the blue path are regarded as real in the opposite case; D gives gradients as a penalty to encourage G to generate more authentic passwords, which is depicted by a red path. This learning framework forces D to achieve stronger judgment criteria than the standard GAN.

Algorithm 1 summarizes the learning procedure for the above described RNN-based GAN. It mainly originates from IWGAN but uses the loss function of a relativistic average GAN. Most notations follow Equation (3). The primary differences are the loss function in lines 9 and 15. These loss functions depend on the relativistic discriminator, estimating the critic for one type of passwords over the average critic of the opposite type. This direct comparison allows G to quickly converge to an optimum point and produce high-quality fake passwords. Also, since our RNN-based GAN adopts IWGAN, we add the gradient penalty to the discriminator's loss function. This penalty forces a 2-norm of gradients for $\hat{x}$ to be less than 1, where $\hat{x}$ is a random sample on a straight line between the pair of points (x_r, x_f). This gives great stability to the training of the GAN. Another important factor is the iteration for optimizing G from lines 13 to 16. Although most GANs have a loop for optimizing D on a given G, this is insufficient to maximize the GAN's performance.

So, we add the loop for training G to stabilize and enhance our GAN. According to our experiments, described in Section 5, this factor has a critical effect on the cracking performance of REDPACK. In general cases, once the model is optimized, the generator is used as a generative model. Instead, we utilize the discriminator as a genuine password estimator for REDPACK.

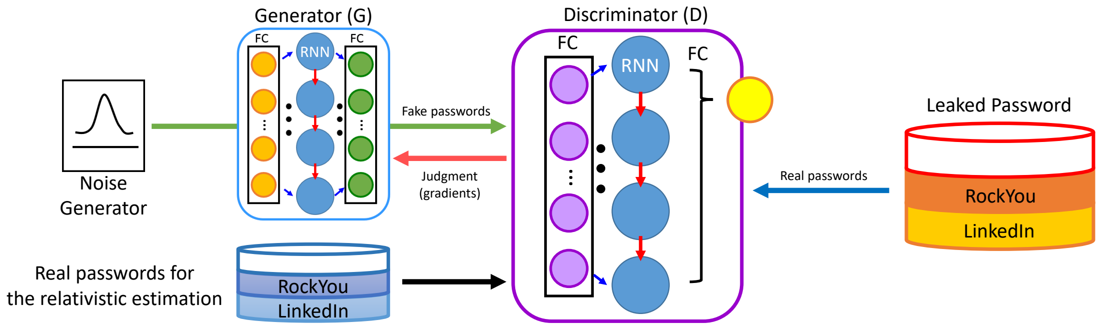


Figure 2. REDPACK Training Phase Diagram.

Algorithm 1 The learning procedure for the RNN-based GAN.

```
Require: Gradient penalty coefficient $\lambda$, number of critic iterations per generator $n_{critic}$, number of
    generator iterations per discriminator $n_{gen}$, batch size $m$, and Adam hyper-parameters $\alpha$, $\beta_1$, and $\beta_2$.
    Critic $C$'s parameters $w$ and generator $G$'s parameter $\theta$. Random samples $\hat{x}$ on straight line between
    real passwords $x_r$ and fake passwords $x_f$.
1:  while $\theta$ has not converged do
2:      for $t = 1, ..., n_{critic}$ do
3:          for $i = 1, ..., m$ do
4:              Sample real data $x_r \sim \mathbb{P}_{data}$, latent variable $z \sim \mathbb{P}_z$, and a random number $\epsilon \sim \mathcal{U}[0,1]$.
5:              $x_f \leftarrow G_\theta(z)$
6:              $\hat{x} \leftarrow \epsilon x_r + (1-\epsilon)x_f$
7:              $\tilde{D}(x_r) = sigmoid(C_w(x_r) - \mathbb{E}[C_w(x_f)])$
8:              $\tilde{D}(x_f) = sigmoid(C_w(x_f) - \mathbb{E}[C_w(x_r)])$
9:              $L_D^i \leftarrow -\mathbb{E}[\log(\tilde{D}(x_r))] - \mathbb{E}[\log(1-\tilde{D}(x_f))] + \lambda\mathbb{E}[(\|\nabla_{\hat{x}} C_w(\hat{x})\|_2 - 1)^2]$
10:         end for
11:         $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m}\sum_{i=1}^m L_D^{(i)}, w, \alpha, \beta_1, \beta_2)$
12:     end for
13:     for $t = 1, ..., n_{gen}$ do
14:         Sample a batch of latent variable $\{z^{(i)}\}_{i=1}^m \sim \mathbb{P}_z$
15:         $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m}\sum_{i=1}^m(-\mathbb{E}[\log \tilde{D}(G_\theta(z_i))] - \mathbb{E}[\log(1-\tilde{D}(x_r))]), \theta, \alpha, \beta_1, \beta_2)$
16:     end for
17: end while
```

4.3. Password Candidates Selecting Structure

The REDPACK consists of multiple password-guessing models: for example, Hashcat, PCFG, rPassGAN with WGAN-GP, and rPassGAN with RaGAN-GP. Two rPassGAN with different loss functions can be used with different hyperparameter configurations as a component of multiple password candidate generators. This is because each RNN-based PassGAN has its own password cracking results that other models could not crack. In our previous research [8,12], both single discriminator rPassGAN and dual discriminator rPassGAN had their own cracked password candidates. So, various deep learning-based password guessing models are used as password candidate sources for REDPACK, as shown in Figure 3. One billion candidates, which are generated by each password generating model, are transformed from strings to tensors. These tensor inputs are supplied to the discriminator (*D*). The discriminator consists of two fully connected neural networks and one layer of an RNN. The RNN uses GRU cells. The discriminator (*D*) estimates how realistic each password input is and provides the probability of each tensor input as an estimated result. The MAX Probability Selector in Figure 3 chooses the password candidate with the highest probability and transforms the tensor into a password string. Then, these selected candidates are transferred to Hashcat or saved in the refined password candidates dictionary. Then, the final step is Hashcat password cracking, as shown in Figure 3. Hashcat's mode is a hybrid attack using transformation rules like the best64, dive, and Rockyou-30000 rules.

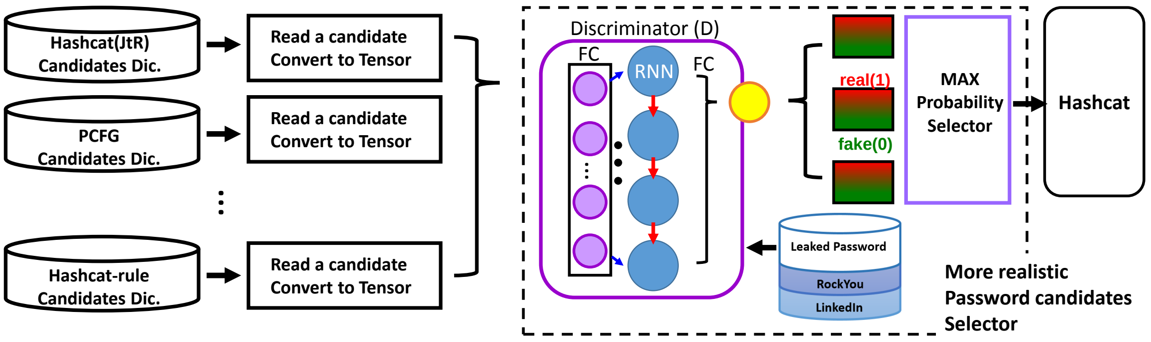

Hashcat(JtR) Candidates Dic.
PCFG Candidates Dic.
Hashcat-rule Candidates Dic.
Read a candidate Convert to Tensor
Read a candidate Convert to Tensor
Read a candidate Convert to Tensor
Discriminator (D)
FC
RNN
FC
real(1)
fake(0)
MAX Probability Selector
Hashcat
Leaked Password
RockYou
LinkedIn
More realistic Password candidates Selector

Figure 3. REDPACK Selecting Phase Diagram.

5. Evaluation

In this section, we explain the configuration of our experiment in detail. Then, we comparatively evaluate the cracking performance and password estimation performance of our approach.

5.1. Experimental Data Preparation

In various studies [2,8,12,14,22,32], a significant amount of leaked passwords have been analyzed. These have notably provided insight into the usage patterns of real-world users. For the experiments in this study, we used publicly available cracked plaintext passwords and leaked password dictionaries. In most of the previous studies on password cracking, leaked Rockyou and LinkedIn passwords were used [2,8,9,12,22,24]. However, our team used additional password dictionaries containing long and 4class passwords for our cracking performance experiments. We use the password classification of Melicher et al. [27]. Melicher et al. defined test datasets by 1class, 2class, 3class, 4class. 2class: passwords in this dataset must contain at least two character classes, 3class: passwords must consist of at least three character classes. 4class: password must contain all classes(symbols, digits, lower and upper letters). Rockyou and LinkedIn contained a few 4class passwords. We also used four cracked password dictionaries from Hashes.org, where several cracked and leaked plaintext passwords were provided. A total of seven training and cracking datasets were present. With these datasets, password cracking performance could be observed in relation to each password length. Information on these datasets is summarized in Table 1. Dataset 1 has been used in several passwords cracking studies. Therefore, through experiments with dataset1, it was possible to compare the performance between our approach and previous studies. With datasets 2–7, we show the cracking performance of our approach in practical situations. Hashcat-Rockyou30000, Hashcat-dive, OMEN, PCFG, and rPassGAN-WGAN were used as the password generators. Subsequently, we applied the best64 rule to all the models' password candidates to maximize cracking performance. This is the method typically used in practice. Each generator model produced one billion password candidates for repeated experiments.

Table 1. Experimental Dataset Summary: we used the plaintext passwords from Hashes.org. 4class set consists of symbols, digits, and upper- and lowercase letters.

Id	Length	Training No.	Cracking No.	Class	Sources
1	1–10	9,421,713	2,481,871	1class	Rockyou
2	11–16	561,552	140,405	4class	Rockyou, LinkedIn, iMesh, Zoosk, Myspace
3	11	224,328	56,083	4class	Rockyou, LinkedIn, iMesh, Zoosk, Myspace
4	12	153,912	38,478	4class	Rockyou, LinkedIn, iMesh, Zoosk, Myspace
5	13	92,156	23,039	4class	Rockyou, LinkedIn, iMesh, Zoosk, Myspace
6	14	55,432	13,858	4class	Rockyou, LinkedIn, iMesh, Zoosk, Myspace
7	15	27,341	6836	4class	Rockyou, LinkedIn, iMesh, Zoosk, Myspace

5.2. REDPACK Training Configuration

To optimize the deep learning model's performance, it is essential to define the training hyper-parameters' values. Unlike other deep learning models, GAN has the G/D training iteration as its specific hyper-parameter. This parameter affects training stability and performance. The essential training hyper-parameters for our experiments are as follows.

- The G/D iteration number represents the number of generator and discriminator training iterations. Although 1:1 and 1:10 are typically used for the discriminator in RaSGAN, 40:10 is usually used for WGAN-GP.
- The batch size represents the number of passwords from the training set that are transferred to the model at each step of the optimization. Although higher values improve the generality of the model, they can result in unstable training. Thus, using the optimal value is important. Typically, a batch size of 128 is used. A batch size of 64 is used to counter training instability.
- An epoch represents the duration of a phase of model training. Longer epochs result in the model fitting the training data better. Although increasing the number of training datasets can enhance the model's performance, it may worsen the general performance of a trained model.

All the parameters are summarized in Table 2.

Table 2. REDPACK Hyper-parameters.

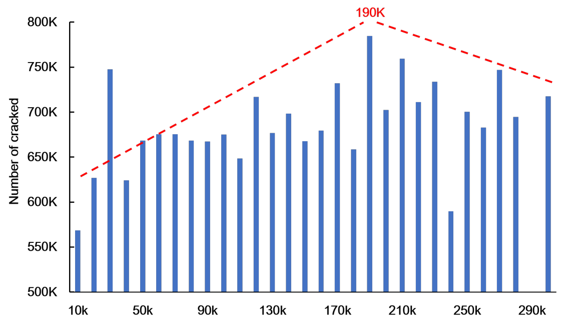
Parameter Name	Value
G/D Training Ratio	RaGAN: 1:1, 1:10 WGAN-GP: 40:10, 20:5, 1:10
Batch Size	Train phase: 128 Train phase(unstable): 64 Select phase: 5000
Training Epochs	100,000, 200,000(WGAN-GP)
GP Lambda	10
Learning Rate	0.0001
Adam opt. β_1	0.5
Adam opt. β_2	0.9

We used Tensorflow-gpu 1.10.1 with Python version 3.5.4 for GPU computing. All the experiments were conducted on the OpenHPC system that NMLab of Korea Univ. developed. Each node of OpenHPC runs on a CentOS 7 server with 32GB Memory; the nodes use Intel Xeon E5 2.20GHz CPUs(x2) and Nvidia TitanXP 12GB GPUs(x4).

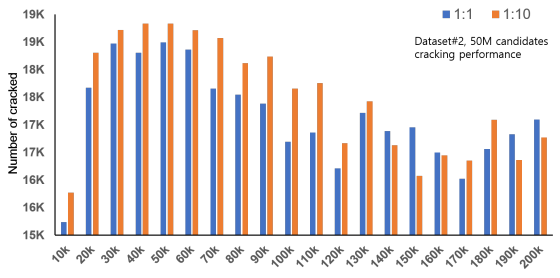
5.3. GAN Training and Testing

As inferred from our previous research [8,12], the major factors that determine cracking performance are the epoch, G/D ratio, and recurrent neural network (RNN) cell type. In the case of the training epoch, models trained for too many epochs are prone to overfitting, as shown in

Figure 4a. Overfitting should be avoided to maximize the effectiveness of selecting realistic password candidates. The G/D ratio also has a powerful effect on the cracking performance. To improve the cracking performance, it is necessary to apply a wide range of G/D ratios to the model. However, this is time-consuming. Therefore, we conducted experiments using G/D ratios of 1:1 and 1:10 for the RaSGAN-GP cost function, as shown in Figure 4b. Although neither may be the optimal value, they suffice to show the effectiveness of the model. The final factor to consider is the cell type. It is necessary to determine which is better between Long Short-Term Memory (LSTM) [28] and Gated Recurrent Unit (GRU) [33] for the RNN-based relativistic discriminator. Throughout experiments for testing these factors, 200k training epochs, a 1:10 G/D ratio, and the GRU cell type were settled on as the optimal settings for password cracking and to ensure generality, as shown in Figure 5.

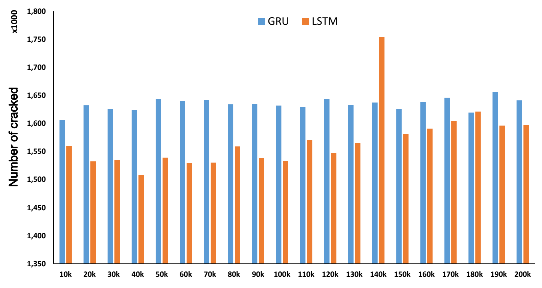


(**a**) Configuration Training Epoch

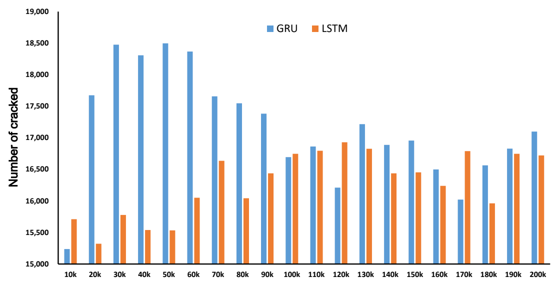


(**b**) Configuration G/D Ratio

Figure 4. Cracking performance per training epoch: After 190k epochs, the password cracking performance was degraded. Training for too many epochs has a bad effect on relativistic discriminator. Based on the number of cracked passwords, G/D ratio of 1:10 yielded better password cracking performance. We could not ascertain whether 1:10 was the best value or not. However, G/D ratio of 1:10 is better than 1:1, which was proposed by Jolicoeur-Martineau [13].



(**a**) LSTM vs. GRU(Dataset1)

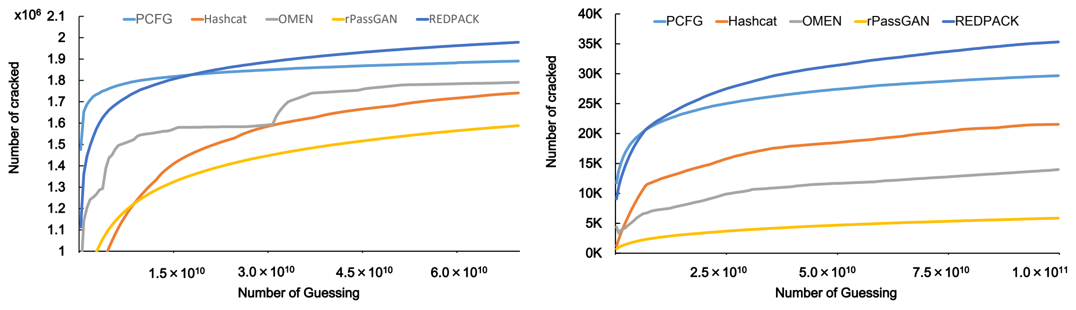


(**b**) LSTM vs. GRU(Dataset2)

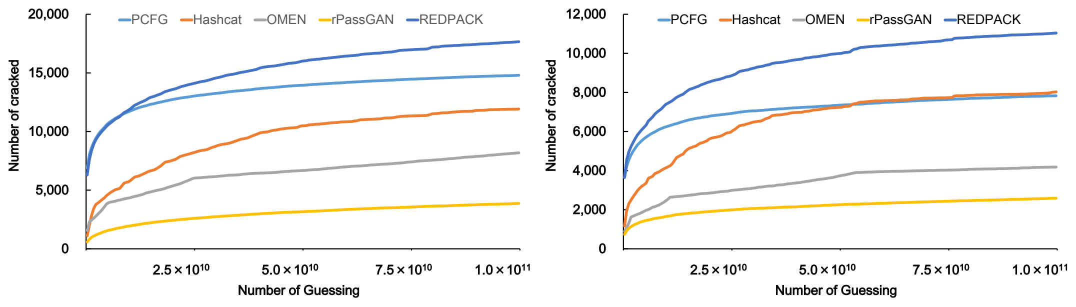
Figure 5. The GRU cell performed better than LSTM. Although LSTM exhibited a lower JSD value during training, it was outperformed by GRU, with respect to generality.

5.4. Password Cracking

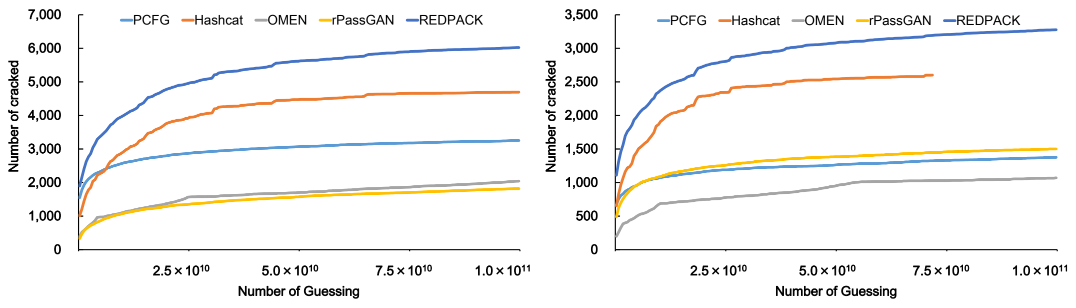
In this section, we present the results of several experiments. Multiple datasets, which are mentioned in Table 1, were used in these experiments. Our model, REDPACK, outperformed all other models at password cracking across all the experiments. First, dataset 1 (a short length Rockyou dataset) was used. This dataset has been used in many previous passwords cracking studies [2,8,9,12,14,24]. Figure 6 shows a cracking performance comparison between REDPACK and other password candidate generators. The x-axis shows the number of password guesses (or the used password candidates). The y-axis is the number of cracked target passwords. In the experiment with dataset 1, all the password-guessing models exhibited similar performance. REDPACK cracked 5%(87,946) more passwords than the PCFG. Although the short Rockyou dataset test was sufficient for theoretic performance comparison, it does not reflect recent password usage trends.



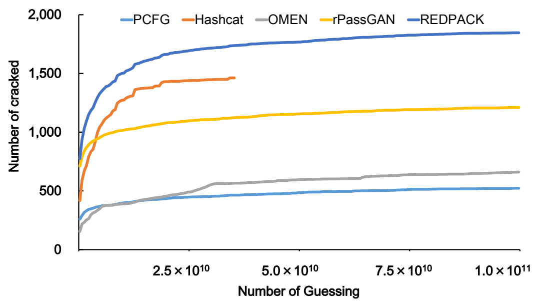
Dataset 1(1–10, 1class) & Dataset 2(11–16, 4class)



Dataset 3(11, 4class) & Dataset 4(12, 4class)



Dataset 5(13, 4class) & Dataset 6(14, 4class)



Dataset 7(15, 4class)

Figure 6. REDPACK exhibits better password cracking performance than any single password generation model.

For a practical password cracking comparison, datasets 2–7 were created and used. Multiple experiments were conducted repeatedly. These experiments used only 4class passwords for a comparison of extremely difficult password cracking. First, Hashcat (best64, rockyou-30000, and dive), PCFG, and rPassGAN were used for the password generation unit. Overall, REDPACK showed 5–20% better cracking performance than any single password guessing model on dataset2 to dataset7. In the case of dataset1 (short-length password), PCFG showed strong performance in the early cracking stages. In the second half of cracking, REDPACK eventually overtook PCFG. However, as password-length gets longer (from dataset3 to dataset7), REDPACK outperformed PCFG and other password candidate generators clearly. That is to say, the effectiveness of REDPACK was shown clearly with complicated and long-length passwords. In the case of datasets 6 and 7, our team could not create one billion Hashcat candidates because of the small amount of training data. Rule-based candidate generation was defined as the multiplication of the number of rules and the number of candidates.

In Table 3, the number of each model's candidate selected by the MAX Probability Selector from the Figure 3 was proportional to the single model's password cracking performance. This result shows that REDPACK did not select candidates randomly but chose candidates selectively. Additionally, this means that the discriminator of REDPACK correctly evaluates the probability of how realistic passwords are generated.

Table 3. Number of candidate of each model consisting of REDPACK candidates. Number in parenthesis denotes cracking performance rank among input sources.

Dataset Id	Hashcat	PCFG	rPassGAN
1	388,369,672 (2)	375,427,872 (1)	236,202,456 (3)
2	391,559,411 (2)	405,053,223 (1)	203,387,366 (3)
3	348,230,515 (1)	426,854,183 (2)	224,915,302 (3)
4	453,200,929 (1)	369,617,876 (2)	177,181,195 (3)
5	533,483,250 (1)	288,233,760 (2)	178,282,990 (3)
6	39,213,755 (1)	455,574,382 (3)	505,211,863 (2)
7	19,847,837 (1)	419,593,769 (3)	560,558,394 (2)

5.5. Limitation of REDPACK

REDPACK showed better performance than any single guessing model. However, its cracking performance was also limited. The relativistic discriminator apparently selected more realistic password candidates. However, more realistic password candidate selection does not always guarantee effective password cracking. Although REDPACK compressed the number of password candidates by up to 66% in the case of three generators, it also missed some candidates that could be important to password cracking. Table 4 shows the performance loss of REDPACK. In the experiment, we included OMEN as a password candidate generator component. The inclusion of OMEN as a password generating unit worsened the cracking performance of the candidate dictionary by REDPACK. OMEN prevented the discriminator from selecting PCFG candidates. This degradation was caused by REDPACK's incorrect selections. OMEN hindered the selection of the PCFG candidates, which could potentially crack the password. Both OMEN and PCFG have similar characteristics. They both generate password candidates with high-order probabilities. To make up for the loss of these two probability based-models, we simply apply a random shuffle to the password candidate sets from both OMEN and PCFG. This simple approach cannot remove the loss of cracking performance completely.

When models based on three different approaches (Hashcat: rule-based, PCFG: probability-based, rPassGAN: deep learning-based) were used as generators, REDPACK was at its most efficient in our experiments.

Table 4. Performance loss of REDPACK was compared with the union set of unit generator models' cracking results. When there were three unit models, REDPACK consisted of PCFG, Hashcat (dive, rockyou-30000), when there were four it consisted of rPassGAN, and PCFG, OMEN, Hashcat and rPassGAN.

Dataset ID	Number of Generators	REDPACK	Union Set Being Cracked
1	3	1,978,429 (−6.7%)	2,119,814
	4	1,995,126 (−6.4%)	2,132,173
2	3	35,699 (−19.2%)	44,200
	4	35,344 (−24.9%)	46,495
3	3	17,659 (−18.9%)	21,801
	4	17,378 (−26.4%)	22,655
4	3	11,039 (−17.4%)	13,375
	4	10,464 (−25.2%)	13,859
5	3	6022 (−17.4%)	7293
	4	5754 (−21.8%)	7505
6	3	3277 (−16.6%)	3933
	4	3128 (−22.2%)	4024
7	3	1846 (−12%)	2128
	4	1798 (−16.8%)	2163

5.6. Further Improvement of Cracking Performance Using Proper Rules

Throughout the password cracking experiments, the best64 rule was applied for all the datasets. The best64 rule is considered the most efficient ruleset because it has been updated through cracking competitions held by the Hashcat team. In the practical field, the best64 rule is used in the first cracking step. However, it was not the optimal rule set for REDPACK. Therefore, to further improve REDPACK's cracking performance, seven Hashcat rules (best64, dive, specific, generated, InsidePro-PasswordPro, Incisive-leetspeak, and T0X1Cv1) were combined into a huge rule file. Then, the number of chances to contribute to password cracking based on each rule for Dataset 2 was noted. For the custom Hashcat rule set of REDPACK$_{U4}$, 100 rules(the same amount as best64) were chosen. REDPACK$_{U4}$ consists of Hashcat, PCFG, OMEN, and rPassGAN. REDPACK$_{U3}$ consists of Hashcat, PCFG, and rPassGAN. Our team evaluated the generality of this custom set of rules using the other datasets, 3, 4, 5, 6, 7 with REDPACK$_{U3}$. The candidates of REDPACK$_{U3}$ were distinct from those of REDPACK$_{U4}$. Furthermore, we created another dataset to establish the generality of the test. The dataset was based on LinkedIn passwords; the length of the dataset, which was composed of 4class (symbol, digits, lower and upper letter) passwords, was from 8 to 10. The ID of this dataset was 1-1. In Table 5, the efficiency of this REDPACK rule was higher than that of best64 in the case of the REDPACK candidates. When the REDPACK rule was applied to PCFG, the REDPACK rule was as effective as best64. This custom set of rules can be improved progressively through various cracking experiments.

Table 5. Cracking performance of custom rule set for REDPACK; this custom set of rules was effective when applied to REDPACK. When this custom rule was applied to both the PCFG and OMEN, cracking performance of custom set of rules was less than that of best64 in some cases. Number in parentheses shows percentage increase or decrease in relation to best64.

Dataset ID	Rule	REDPACK$_{U3}$	PCFG	OEMN	rPassGAN
1-1	best64	69,473	69,133	53,224	33,278
	custom	72,387 (4%)	66,793 (−3.8%)	55,391 (4%)	35,323 (6%)
2	best64	35,699	14,796	8184	5858
	custom	43,442 (22%)	16,014 (8%)	11,106 (36%)	8568 (46%)
3	best64	17,475	14,796	8184	3871
	custom	22,042 (26%)	16,013 (8.2%)	11,106 (35%)	6427 (66%)
4	best64	10,819	7830	4185	2588
	custom	13,309 (23%)	8482 (8%)	5179 (23%)	3077 (19%)
5	best64	5984	3253	2042	1819
	custom	7058 (17%)	3558 (9%)	2369 (16%)	2071 (14%)
6	best64	3252	1376	1070	1500
	custom	3728 (14%)	1491 (8%)	1145 (7%)	1555 (4%)
7	best64	1803	523	661	1210
	custom	2050 (13%)	565 (8%)	671 (1.5%)	1233 (2%)

6. Conclusions

In this paper, we proposed a deep learning-based method to build an effective password cracking dictionary from multiple models' password candidates. Our model is particular in that it is the first approach to apply a GAN discriminator to the password cracking domain. Generally, because GANs are a generative deep learning model, the generator of the GAN is used. However, in our model (REDPACK), the discriminator of the GAN is a core component. For accomplishing the goal of making the discriminator of the GAN choose more realistic password candidates, we adopt the RaSGAN-GP cost function, which was proposed by Jolicoeur [13]. Through several password-cracking experiments, we showed that the deep learning-based password candidate selector, REDPACK, outperformed every other individual password candidate-guessing model (Hashcat, PCFG, OMEN, and rPassGAN) in password cracking performance experiments. These experiments also showed the effectiveness of the GAN's discriminator. Finally, we demonstrated that our custom set of rules for REDPACK improved its cracking performance, compared to ready-made rules such as the best64. Significantly, this set of rules were successfully applied to REDPACK and rPassGAN. However, in the case of PCFG and OMEN, this set of rules worsened the cracking performance compared to the best64. Although there are some defensive studies [34–37] that counter offline dictionary attacks, REDPACK, by facilitating the creation of a small and effective password cracking dictionary, enabled us to execute a fast attack. From a security perspective, the outperformed cracking model's candidates can enhance password strength estimators, such as Zxcvbn [7], by filtering the guessable password candidates from our previous study [12]. Therefore, it is clear that the candidates produced by REDPACK can fortify the password strength estimator. REDPACK can be used for both attacking and defending the security of passwords. Through the rPassGAN [8,12] and REDPACK studies, we achieved a new way to improve password cracking performance by incorporating both the generator and discriminator of a GAN. In the future, we will continue to conduct research into improving the performance of REDPACK.

Author Contributions: Conceptualization, S.N.; methodology, S.N.; software, S.N., S.J.; validation, S.N, S.J.; data curation, S.N.; writing—original draft preparation, S.N.; writing—review and editing, J.M.; supervision, J.M.; All authors have read and agreed to the submitted version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: This research was supported by a Korea University Grant.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CNN	Convolutional Neural Network
GAN	Generative Adversarial Networks
GRU	Gated Recurrent Unit
JtR	John the Ripper
JSD	Jensen-Shannon Divergence
LSTM	Long-Short Term Memory
PCFG	Probability Context-Free Grammar
OMEN	Ordered Markov ENumerator
REDPACK	Relativistic Discriminator for Password Candidates Effective Pack
RNN	Recurrent Neural Network
rPassGAN	Recurrent PassGAN

References

1. Dell' Amico, M.; Michiardi, P.; Roudier, Y. Password Strength: An Empirical Analysis. In Proceedings of the 2010 Proceedings IEEE INFOCOM, San Diego, CA, USA, 14–19 March 2010; pp. 1–9. [CrossRef]
2. Dürmuth, M.; Angelstorf, F.; Castelluccia, C.; Perito, D.; Chaabane, A. OMEN: Faster Password Guessing Using an Ordered Markov Enumerator. In Proceedings of the International Symposium on Engineering Secure Software and Systems, Milan, Italy, 4–6 March 2015.
3. Ma, W.; Campbell, J.; Tran, D.; Kleeman, D. Password Entropy and Password Quality. In Proceedings of the 2010 Fourth International Conference on Network and System Security, Victoria, Australia, 1–3 September 2010; pp. 583–587.
4. John the Ripper Password Cracker. Available online: http://www.openwall.com/john/ (accessed on 12 July 2018).
5. Hashcat Advanced Password Recovery. Available online: https://hashcat.net/wiki/ (accessed on 12 July 2018).
6. Hashcat Rules. Available online: https://github.com/hashcat/hashcat/tree/master/rules (accessed on 12 July 2018).
7. Daniel Lowe, W. zxcvbn: Low-Budget Password Strength Estimation. In Proceedings of the 25th {USENIX} Security Symposium ({USENIX} Security 16), Austin, TX, USA, 10–12 August 2016; {USENIX} Association: Berkeley, CA, USA, 2016; pp. 157–173.
8. Nam, S.; Jeon, S.; Moon, J. A New Password Cracking Model with Generative Adversarial Networks. In *Information Security Applications*; You, I., Ed.; Springer International Publishing: Cham, Switzerland, 2020; pp. 247–258, [CrossRef]
9. Hitaj, B.; Gasti, P.; Ateniese, G.; Perez-Cruz, F. PassGAN: A Deep Learning Approach for Password Guessing. *arXiv* **2017**, arXiv:1709.00440.
10. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Networks. In *Advances in Neural Information Processing Systems 27*; Curran Associates, Inc.: Montreal, QC, Canada, 8–13 December 2014; pp. 2672–2680.
11. Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; Courville, A.C. Improved Training of Wasserstein GANs. In *Advances in Neural Information Processing Systems 30*; Curran Associates, Inc.: Long Beach, CA, USA, 4–9 December 2017; pp. 5767–5777.
12. Nam, S.; Jeon, S.; Kim, H.; Moon, J. Recurrent GANs Password Cracker For IoT Password Security Enhancement. *Sensors* **2020**, *20*, 3106. [PubMed]
13. Jolicoeur-Martineau, A. The relativistic discriminator: A key element missing from standard GAN. *arXiv* **2018**, arXiv:1807.00734.
14. Weir, M.; Aggarwal, S.; Collins, M.; Stern, H. Testing metrics for password creation policies by attacking large sets of revealed passwords. In Proceedings of the ACM Conference on Computer and Communications Security, Chicago IL, USA, 4–8 October 2010; ACM Press: New York, NY, USA, 2010; pp. 162–175. [CrossRef]
15. Ma, J.; Yang, W.; Luo, M.; Li, N. A Study of Probabilistic Password Models. In Proceedings of the 2014 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 18–21 May 2014; pp. 689–704.

16. Bidgoli, H. Handbook of Information Security, Threats, Vulnerabilities, Prevention, Detection, and Management. In *Handbook of Information Security*; Wiley: Hoboken, NJ, USA, 2006.
17. KoreLogic's Rules in John the Ripper. Available online: https://contest-2010.korelogic.com/rules.html (accessed on 12 July 2018).
18. Hashcat Per Position Markov Chains. Available online: https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/hashcat-per-position-markov-chains/ (accessed on 12 July 2018).
19. JTR Markov Generator. Available online: https://openwall.info/wiki/john/markov (accessed on 12 July 2018).
20. Hashes.org Leaked Hashes. 2020. Available online: https://hashes.org/leaks.php (accessed on 5 October 2018)
21. Narayanan, A.; Shmatikov, V. Fast dictionary attacks on passwords using time-space tradeoff. In Proceedings of the 12th ACM Conference on Computer and Communications Security–CCS 05, Alexandria, VA, USA, 7–11 November 2005.
22. Weir, M.; Aggarwal, S.; de Medeiros, B.; Glodek, B. Password Cracking Using Probabilistic Context-Free Grammars. In Proceedings of the 2009 30th IEEE Symposium on Security and Privacy, Oakland, CA, USA, 17–20 May 2009; pp. 391–405.
23. Yazdi, S.H. Probabilistic Context-Free Grammar Based Password Cracking: Attack, Defense and Applications. Ph.D. Thesis, Department of Computer Science, Florida State University, Tallahassee, FL, USA, 2015.
24. Houshmand, S.; Aggarwal, S.; Flood, R. Next Gen PCFG Password Cracking. *IEEE Trans. Inf. Forensics Secur.* **2015**, *10*, 1776–1791. [CrossRef]
25. Houshmand, S.; Aggarwal, S. Using Personal Information in Targeted Grammar-Based Probabilistic Password Attacks. *IFIP Adv. Inf. Commun. Technol.* **2017**, *511*, 285–303. [CrossRef]
26. National Institute of Standards and Technology. Digital Identity Guidelines. 2004. Available online: https://pages.nist.gov/800-63-3/ (accessed on 10 June 2017).
27. William, M.; Blase, U.; Sean M., S.; Saranga, K.; Lujo, B.; Nicolas, C.; Lorrie Faith, C. Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks. In Proceedings of the 25th {USENIX} Security Symposium ({USENIX} Security 16), Austin, TX, USA, 10–12 August 2016; pp. 175–191.
28. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
29. Arjovsky, M.; Chintala, S.; Bottou, L. {W}asserstein Generative Adversarial Networks. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; Volume 70, pp. 214–223.
30. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
31. Shang, W.; Chiu, J.; Sohn, K. Exploring Normalization in Deep Residual Networks with Concatenated Rectified Linear Units. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; pp. 1509–1516.
32. Castelluccia, C.; Dürmuth, M.; Perito, D. Adaptive password-strength meters from Markov models. In Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 21–24 February 2012.
33. Cho, K.; van Merrienboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014; Association for Computational Linguistics: Stroudsburg, PA, USA, 2014; pp. 1724–1734. [CrossRef]
34. Bellare, M.; Pointcheval, D.; Rogaway, P. Authenticated Key Exchange Secure against Dictionary Attacks. In Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques, Bengaluru, India, 5–13 December 2000; Springer: Berlin/Heidelberg, Germany, 2000; pp. 139–155.
35. Bellovin, S.M.; Merritt, M. Encrypted Key Exchange: Password-Based Protocols SecureAgainst Dictionary Attacks. In Proceedings of the 1992 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 4–6 May 1992; p. 72.

36. Lucks, S. Open key exchange: How to defeat dictionary attacks without encrypting public keys. *Lect. Notes Comput. Sci.* **1998**, *1361*, 79–90. [CrossRef]
37. Katz, J.; Ostrovsky, R.; Yung, M. Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords. In Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology, Innsbruck, Austria, 6–10 May 2001; Springer: Berlin/Heidelberg, Germany, 2001; pp. 475–494.