Modeling Human Tendencies for Password Guessing

Rohit Mutalik, Dhairya Chheda, Zeeshan Shaikh, and Dhanashree Toradmalle

Abstract Passwords form a predominant part of all authentication mechanisms and likely will not be replaced soon, primarily because of their ease of use and straightforwardness. Human-generated passwords are particularly susceptible to guessing attacks as a consequence of the limitation of precise recall and hence are not random. In this paper, we make use of the excellent expressive power of sequence modeling neural networks such as LSTMs and GRUs to effectively guess passwords as compared to other cutting-edge password guessing techniques like Markov models, JtR, and PCFGs. LSTM and GRU models were able to match about 55% of the testing set when evaluated on the Rockyou dataset. This proves that sequence modeling neural networks can effectively learn the distribution of real passwords from previously leaked datasets.

Keywords Long short-term memory · Gated recurrent unit · Generative adversarial networks · Password guessing · John the ripper · Hashcat · Password analysis · IWGAN

1 Introduction

Password-based authentication, even with its glaring deficiencies, shows no signs of being replaced owing to its excellent balance of security and ease of use. Human limitations to recall and password-based authentication mechanisms requiring precise

R. Mutalik (✉) · D. Chheda · Z. Shaikh · D. Toradmalle
Shah and Anchor Kutchhi Engineering College, Mumbai, India
e-mail: rohit.mutalik@sakec.ac.in

D. Chheda
e-mail: dhairya.chheda@sakec.ac.in

Z. Shaikh
e-mail: zeeshan.shaikh@sakec.ac.in

D. Toradmalle
e-mail: dhanashree.toradmalle@sakec.ac.in

M. Tuba et al. (eds.), *ICT Systems and Sustainability*, Lecture Notes in Networks and Systems 321, https://doi.org/10.1007/978-981-16-5987-4_26

recall lead to the selection of short [1] human-memorable passwords and their repeated use for authentication on multiple different security systems. Several web applications and security systems proactively tackle security risks arising from weak passwords by requiring users to use a combination of several different types of characters such as special characters and numerals and using password guessing tools such as Hashcat and JtR to find weak passwords.

Secure systems requiring passwords for authentication are susceptible to guessing attacks if security measures such as rate-limiting and salting are improperly implemented, as in the case of SolarWinds Breach [2]. Hackers can also hash probable passwords and search for matches and cracked hashes password hash database dumps found online. Learning the password set distribution allows us to sample the least probable password sequences and thus evade the aforementioned attacks instead of relying on static strength evaluation rules implemented by most websites.

In the case of offline guessing attacks, where the password p has to be recovered when the hash h is known, rainbow tables [3] can be used. Oechslin's rainbow table attack makes use of a reduction function R that performs a mapping between the hash function H and the set of passwords P. Alternate calls of H and R are used to generate a single chain. This method requires low disk storage space; however, it suffers from false alarm where h is not in the generated chain even when extended h matches with the endpoint of the chain. The password cracking performance of rainbow tables also heavily depends on the reduction function R chosen.

Traditional password guessing tools such as Hashcat and JtR utilize two approaches. A brute-force attack is the first solution, whereas a dictionary attack and variations are the second.

Brute-force attacks focus on constructing a candidate a password generated by combining several different types of characters while taking into consideration, the minimum and maximum lengths of the password and the set of all legal characters. The performance of such attacks has been greatly improved by optimizing the tools to run on graphical processing units (GPU); however, large space of legal characters and long passwords require an enormous amount of time and render such an approach unusable.

Dictionary-based attacks make use of attack dictionaries for password guessing. Previously disclosed password sets and a combination of words from the English dictionary and numbers to generate password guesses. Tools such as Hashcat [4] and JtR [5] utilize several mangling and concatenation rules to generate more candidate password guesses and expand the existing dictionary. Hashcat provides several built-in rulesets such as best64 which reasonably improves the performance of password cracking. These rulesets amplify the existing dictionary by changing character capitalization, appending or prepending numbers and adding special characters to an already existing password in the dictionary. However, such an attack is constrained by the distribution of leaked passwords and the size of the dictionary. This method also struggles with finding rulesets that improve the cracking performance since analyzing statistical patterns of tens of millions of passwords is tedious.

In this research, we suggest the use of excellent expressive power of neural networks to analyze existing statistical patterns in the leaked dataset and sample novel sequences from the modeled distribution to generate password guesses. Approaches that perform probabilistic analysis on the password set are also explored.

2 Background and Related Work

In this section, we describe other password guessing approaches that have been studied previously. The vulnerability of authentication systems to password guessing attacks is extremely conditional upon the type of attack vector as well as the configuration of the system itself. Guessing attacks are rendered useless for systems that lock the user out for a certain duration after a specific number of incorrect guessing attempts.

2.1 Probabilistic Approaches

Markov models were suggested by Narayanan et al. for password guessing suggesting that phonetic similarity of words play an important role in the memorability of passwords [6]. The Markov model estimates the likelihood of the next character occurring by taking previous background characters into account. Given a prefix of length $(n-1)$, an n-gram Markov model predicts the likelihood of the nth character. As a result, for any string $c_1, \ldots, c_t$, probability estimation is done as follows:

$$P(c_1, \ldots, c_t) \approx P(c_1, \ldots, c_{n-1}) \cdot \prod_{i=n}^{t} P(c_i | c_{i-n+1}, \ldots, c_{i-1}) \tag{1}$$

Narayanan et al. also proposed a hybrid algorithm that generated passwords that matched the Markovian filter and are accepted by a finite automaton. However, finding suitable regular expressions by analyzing the password set containing millions of passwords is difficult. This work was subsequently improved by Deurmuth et al. [7] by generating passwords in descending order of probability thus minimizing password cracking time by trying the most probable guesses first.

Weir et al. [8] proposed probabilistic context-free grammars (PCFG) for generating password guesses. PCFGs treat passwords as having a certain grammatical structure and a set of terminals that fit into those structures. Such methods generated preterminal structures by substituting digit string and special string terminals into base structures in order of decreasing probability. Furthermore, most probable candidate password guesses are enumerated first by substituting alpha string terminals into preterminal structures sorted in descending order of probability. The probability of the generated password is calculated by multiplying individual probability of the

base structure, preterminal structure, and the substituted string. In the experiments, PCFG-based password guessing mechanism outperformed JtR operating in wordlist mode.

2.2 Deep Learning Approaches

The use of artificial neural networks to model the password set distribution was proposed by Melicher et al. to gauge the strength of human-generated passwords. FLA primarily aimed at evaluating a password's strength and the number of guesses needed to crack it on the client-side browser while keeping the model as small as possible for achieving sub-second latency [9].

PassGAN [10] was developed by Hitaj et al. which makes use of improved training of Wasserstein Generative Adversarial Network (IWGAN) to generate password samples with probability distribution p_g that is close to the real password distribution p_{real}. Gulrajani et al. [11] proposed IWGAN and introduced gradient penalty as a measure of enforcing 1-Lipschitz Continuity for stable training of WGAN [12]. The goal of generator **G** is to transform the original input noise vector z into password guesses using a series of residual blocks consisting of two 1D convolutions such that the discriminator is unable to differentiate between genuine and counterfeit samples. The discriminator network **D** aims to differentiate between the real passwords sampled from the dataset as well as the counterfeit examples generated by the generator network G and output a score estimating the realness of the generated password. The objective function of GAN [13] is given as follows:

$$\min_{\theta_G} \max_{\theta_D} \sum_{i=1}^{n} \log f(x_i; \theta_D) + \sum_{j=1}^{n} \log\left(1 - f(g(z_j; \theta_G); \theta_D)\right) \tag{2}$$

Further attempt to model the password set distribution using IWGAN was done by Nam et al. in [14]. Here, both the generator and the discriminator were recurrent neural networks. Nam et al. also experimented with a dual discriminator approach yielding slightly better results.

3 System Design

We explored a wide set of parameters to improve the cracking performance and approached password guessing as a sequence modeling task using long short-term memory (LSTM) and gated recurrent unit (GRU) cells.

3.1 Model Design

To avoid the recomputation of recurrent cell states of context characters while outputting the probabilities of the next character for passwords with common prefixes, we use a shared layer architecture where the training and inference models both share the same underlying dense and LSTM layers (see Fig. 2). This allows us to train the model by requiring the entire password but to predict the probability of next character in password sequence, only the previously computed states and last generated character are required. This reduces the amount of computation required for sampling passwords with common prefixes. The training model performs a forward pass over the network to generate the probability distribution over the token space for the next character in the sequence and then performs a backward pass to update weights of the shared layers based on the gradients obtained from loss calculation. The inference model performs only forward passes to generate probabilities during the password sampling process.

3.2 Model Prediction

Neural networks, like Markov models, are taught to anticipate the next character in the password given the state vectors representing all preceding characters. The network, given a sequence of characters, also models the probability of an END token denoting the end of the password. The generation of passwords starts by feeding the model a START token after which the model outputs the probabilities of being the first character over the entire token space. For example, the probability of password 'pwd' is calculated by using the start token (t) as a context and then question the model for the probability of having 'p' as the first character, then the probability of 'w' as the second character after 'p', then the probability of 'd' as the third character after 'pw' and finally the probability of entire password 'pwd' by questioning the model for the probability of end token after 'pwd'. Figure 1 shows the password sampling process.

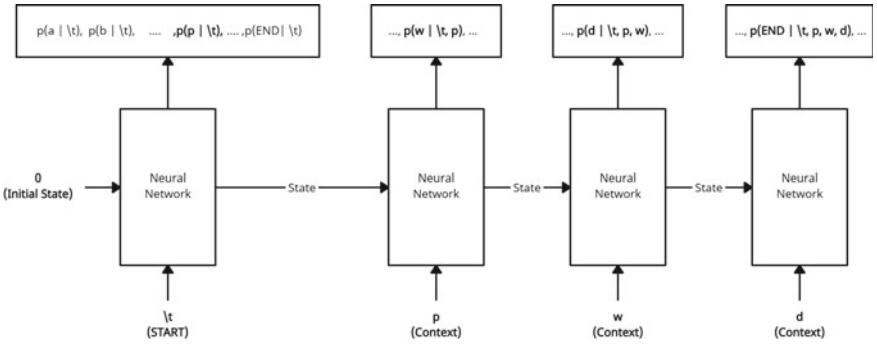

Fig. 1 Password sampling

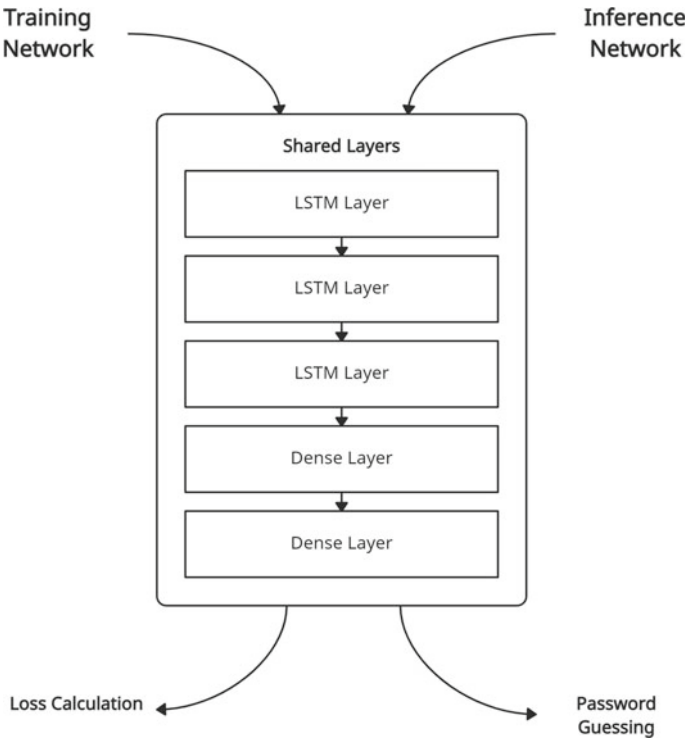

Fig. 2 Shared layer architecture

4 Our Approach

4.1 Token Space

In this work, we focus on a character-level password generation model including all characters, numerals, and special characters. To avoid straining the network by modeling all the characters, as shown empirically by Melicher [9], we choose to only model the probabilities of lowercase characters. Password guesses containing uppercase characters can be generated by post-processing the generated password to generate guesses consisting of both uppercase and lowercase characters based on the relative frequency of occurrence.

4.2 Model Architecture

We utilized the power of recurrent neural networks, which have been shown to process sequential data while maintaining internal state information about previously generated characters in the sequence. Recurrent neural network cells like long short-term memory [15] and gated recurrent units [16] which mitigate the issue of vanishing and exploding gradients and are excellent at generating text from the perspective of character-level language models.

4.3 Teacher Forcing and Exposure Bias

In this work, we implement teacher-forcing to train the neural network while taking into account both the internal memory state information and the previous timestep's ground truth context character. With this approach, the network can suffer from exposure bias during inference where the model can produce substandard sequences due to the propagation of errors. This happens because the network is trained to output conditional probability for the next character in the sequence, based only on the ground truth context character of the previous timestep and not on the generated errors.

4.4 Password Guess Sampling

To generate guesses using the inference model from the learned password set distribution, we use two approaches. First, we sample all the passwords that have the probability over a certain threshold. However, in this approach, the probability of the entire password reduces quickly as we are multiplying numbers that are less than one. This creates an undesirable effect where it prefers short passwords because of the fewer number of characters. In the second approach, we perform length normalization of the generated passwords and assign a score. This incentivizes the sampling algorithm to generate longer sequences especially considering the lack of passwords with a minimum of 16 characters in the Rockyou dataset. Passwords having a score greater than the threshold are then sampled. The normalization constant α can be tuned as per the performance of the sampling algorithm.

$$\text{score} = \frac{1}{T_y{}^{\alpha}} \sum_{t=1}^{T_y} \log P(y^{<t>} | \text{START}, y^{<1>}, \ldots, y^{<t-1>}) \tag{3}$$

T_y denotes the length of the password.

5 Experimental Setup

5.1 Model Implementation

Our implementation makes use of the Keras Library with TensorFlow 2.3 [17] backend for training both the networks. The training models used three stacked recurrent layers with 300 units followed by two densely connected layers with a total of 4,861,207 parameters. The training and sampling of passwords were done in Python 3.8. The tests were carried out on a computer running Ubuntu 20.04 LTS, a 6 core AMD Ryzen 5 3500 CPU, 16GB of RAM, and an NVIDIA GeForce 2060 Super extreme edition GPU with 8GB of VRAM.

5.2 Training and Testing Configuration

Rockyou Dataset To evaluate the performance of the LSTM and GRU networks and generate candidate password guesses, we trained these models on sets of passwords obtained from the Rockyou password dataset leak [18]. We believe this approach is innocuous, even though the passwords might contain personal information since the dataset is publicly available.

In this experiment, we selected all passwords of length less than or equal to 32 from the Rockyou dataset (about 14 million passwords). Before training the model, as a standard preprocessing step, all the unicode and special ASCII control and extended ASCII characters were removed. Out of the total 14 million passwords, about 70% (9,973,206 passwords) were used to train the LSTM and GRU networks. For evaluation purposes, we calculated the set difference between the training set and the remaining 30% of the passwords (4,126,994 total passwords and 3,921,227 unique passwords) resulting in 3,665,812 passwords that the models have not examined during the training.

Candidate Password Sampling Configuration Once the models are trained, the sampling algorithm generates password guesses with a score greater than the configured threshold. The score threshold and alpha parameters for normalization are mentioned in Table 1. The generated passwords are then post-processed using the steps mentioned in Token Space.

Table 1 Sampling algorithm configuration

Model	Threshold	α
LSTM	−8.465	0.37
GRU	−9.45	0.3275

6 Evaluation

6.1 Evaluating Generated Passwords

To evaluate the performance of the models, we generated about 1.57×10^9 and 1.53×10^9 guesses using the LSTM and GRU model, respectively. Inspection of the generated passwords shows that the model has learned to construct meaningful candidate passwords containing names and a combination of names and numbers such as name followed by 2 or 4 digits indicating the birth year of the user. Our results show that neural networks exhibit exceptional password guessing performance for long and complex passwords. For the evaluation of both the models, the entire testing set was further partitioned into three sets based on the length (1 to 8, 9 to 15, and 16 to 32 characters). Figure 3 highlights the relation between the number of password guesses and the total number of passwords matches across all sets for both the models. Our results show that the GRU model slightly outperformed the LSTM model in the ≤8 character testing set with 56% matches as compared to the LSTM model's 50% matches out of the total 2,250,012 testing passwords. However, for the remaining two sets (9–15 and 16–32) LSTM performs better with 34% and 25% matches, respectively, as compared to the GRU model's 32% and 15% out of the total 1,345,062 and 70,738 passwords in 9–15 and 16–32 sets. The findings of the evaluation are detailed in Table 2.

Both the models generate novel sequences by combining words, numbers, and specific frequently occurring substrings of passwords in the training set such as 'love' or '123'. For instance, the model generated 'loveforever', 'love4ever', and

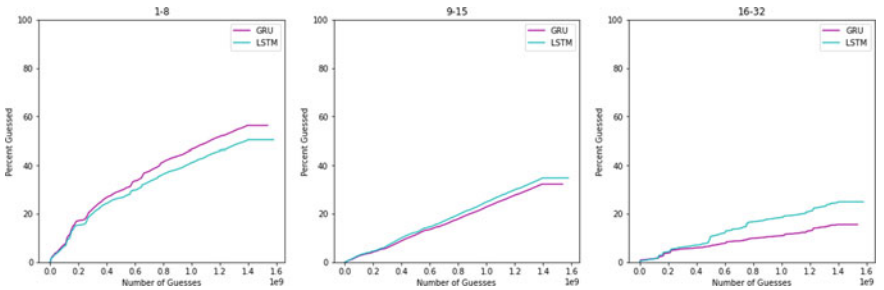

Fig. 3 1–8, 9–15 and 16–32 characters cracking performance

Table 2 Guessing performance of models

Password sets	≤8	≥9 and ≤15	≥16 and ≤32	Total
GRU model	1,266,478 (56%)	433,656 (32%)	10,986 (15%)	1,711,120 (46%)
LSTM model	1,133,612 (50%)	468,565 (34%)	17,628 (25%)	1,619,805 (44%)
Testing set	2,250,012	1,345,062	70,738	3,665,812

Table 3 Sample non-matched passwords

ricardoisthebest	ihatemywife123	ashley1869
@ngelina	idontlikemybabyboy	27/09/06
ilovemyprincess70	strawberrytinkerbell	nokia700
assholeisthebomb	christopher2007@yahoo.com	samurai@123
ilove4everross	teamomuchoasantonio	25aug2007

'love4eva', thus effectively capturing human password generation processes. This enables the models to investigate passwords in areas where conventional cutting-edge rule-based tools cannot. According to our research, the number of cumulative password matches gradually increases with increasing guess attempts. Both the models were able to match more than 40% of the $\leq$8 category passwords at just 10^9 guesses. For targeted password guessing approaches, tools such as common user password profiler (CUPP) can be used to make informed guesses, using commonly seen patterns, Personally Identifiable Information (PII) about the target, such as full name, date of birth, and nickname.

6.2 Non-matched Password Guesses

Closer examination of sampled passwords that were absent in testing set leads us to conclude that these passwords look appropriate for passwords generated by humans. We also believe that, while these passwords did not match with the testing set, they could still match with the passwords chosen by users on websites other than Rockyou. For targeted guessing, the generated passwords can be utilized as templates by replacing names and numbers in the guesses. Although carefully inspecting each generated password for producing targeted guesses is a tedious process, Table 3 lists a small sample of non-matched passwords.

7 Conclusion

We presented two neural networks with LSTM and GRU cells in this paper to learn the distribution of passwords chosen by humans using recently leaked password sets. These models were able to generate passwords similar to humans by exploring a significant portion of the password space without requiring any prerequisite knowledge and analysis of the leaked password dataset. These models have password cracking performance comparable to the Hashcat and JtR, which are cutting-edge password guessing tools even at lower training times.

The models were evaluated on the portions of the dataset that the networks were not trained on, thus determining how well the models were able to figure out how actual passwords are distributed. Our findings reveal that the models were able to generate human-like passwords and matched about 50% of the test set in just 10^9 guesses. The learned distribution can also be used to determine the strength of passwords selected by the user. Suggesting less probable and pronounceable passwords for users by utilizing the learned patterns from leaked passwords remains a high potential research area.

The ubiquitous nature of password-based authentication systems requires that developers adopt secure programming practices and implement password security measures such as two-factor authentication, salting, and limiting the number of incorrect password attempts.

References

1. Morris R, Thomson K, Laboratories B (1979) Password security a case study
2. https://www.zdnet.com/article/cisa-solarwinds-hackers-also-used-password-guessing-to-breach-targets/SolarWindsBreach
3. Oechslin P "Making a Faster Cryptanalytic Time-Memory Trade-Off", Laboratoire de Sécurité et de Cryptographie (LASEC), Ecole Polytechnique Federale de Lausanne, Faculté I&C, 1015 Lausanne, Switzerland
4. https://hashcat.net/hashcat/Hashcat
5. https://www.openwall.com/john/ John The Ripper
6. Narayan A, Shmatikov V (2005) Fast dictionary attacks on passwords using time-space tradeoff. The University of Texas at Austin
7. Duermuth M, Angelstorf F, Castelluccia C, Perito D, Chaabane A (2015) OMEN: faster password guessing using an ordered markov enumerator. In: International symposium on engineering secure software and systems, Mar 2015, Milan, Italy, hal-01112124
8. Weir M, Aggarwal S, de Medeiros B, Glodek B (2009) Password cracking using probabilistic context-free grammars. Department of Computer Science, Florida State University, Tallahassee, Florida 32306, USA (2009)
9. Melicher W, Segreti SM, Komanduri S, Bauer L, Christin N, Cranor LF, Fast, lean, and accurate: modeling password guessability using neural networks. Carnegie Mellon University
10. Hitaj B, Gasti P, Ateniese G, Perez-Cruz F PassGAN: a deep learning approach for password guessing. In: Stevens Institute of Technology, New York Institute of Technology, Stevens Institute of Technology, Swiss Data Science Center (ETH Zurich and EPFL)
11. Ishaan G, Faruk A, Martin A, Vincent D, Aaron C (2017) Improved training of wasserstein GANs. montreal institute for learning algorithms, Courant Institute of Mathematical Sciences
12. Arjovsky M, Chintala S, Bottou L Wasserstein GAN. Courant Institute of Mathematical Sciences, Facebook AI Research
13. Goodfellow IJ, Pouget-Abadie J, Mirza M, Ozair S, Courville A, Bengio Y Generative adversarial nets. Departement d informatique et de recherche op erationnelle, Universite de Montrea, Montreal, QC H3C 3J7
14. Sungyup N, Seungho J, Jongsub M (2020) A new password cracking model. Korea University, Seoul, South Korea, Graduate School of Information Security
15. Hochreiter S, Schmidhuber J Long Short-Term Memory. Technishe Universitat Munchen 80290 Munchen, Germany, Corso, Elvezia 36 6900 Lugano, Switzerland

16. Cho K, van Merrienboer B, Gulcehre C, Bahdanau D, Bougares F, Bengio Y Learning phrase representations using RNN encoder-decoder for statistical machine translation. Universite de Montreal, Jacobs University, Germany, Universite du Maine, France, Universite de Montreal, CIFAR Senior Fellow
17. https://github.com/tensorflow/tensorflow/ Tensorflow Github Repository
18. https://wiki.skullsecurity.org/Passwords Rockyou Dataset