Group Attribute Context-ware Password Guessing for Interactive Applications

C. Lu

***Abstract*—**

***Index Terms*—**

I. Introduction

II. Related Works

III. Group Attribute Password Guessing Scheme

A. Password Dataset Processing

To train the password guessing model, we have collected a huge number of leaked and cracked password sets containing various group attributes. Table I illustrates some of the leaked password sets. Many of these passwords contain personal information about the owner, such as name, birthday, email, etc. In order to protect personal privacy, only the password was used in our experiments. We have selected five group attributes, corresponding to Game, Programmer, Marriage, Writing, Social. In order to represent these attributes, 1-of-N encoding has been used, that is, a five dimensional vector $\boldsymbol{g}_c$, where only the c^{th} dimension is one and the rest are zero. Each of the password set has been divided into three parts: a training set, a validation set and a test set. For the password in the test set we have used MD5 encryption so that the hashcat **??** can be used to evaluate the efficiency of the generated password set by our model. Hashcat is an advanced password recovery tool.

As will be mentioned below, the main idea of the model is using the current password character to predict the next character. To achieve this, the trianing password was represented as matrices. During training, we must specify the length of input sequence in the model, denoted by T in this paper, which is a hyper parameter that needs to be set manually. Various sequence length has been tested and made comparisons, which will be shown in the section IV. Assume that the length of sequence is chosen as 4, considering the password *iloveyou*, the first input of the model is *ilov*, and the corresponding label is *love*. In the next round of training, the input becomes *love*, and the label is changed to *ovey*. In generation phase, for the same password *iloveyou*, the first input character is *i*, which can be represented by an n-dimensional vector, where n refers to the size of alphabet in password set. The model then learns that given the first character, the next character will be *l*. The next time step, the model will loads in two characters, and learns given the sequence *il*, the next character will be *o*. This process is repeated until it generate the *<eos>*, which refers to the end of a password.

TABLE I
Password sets for various group attributes

Password Set	**Type**	**Leak Method**	**Total Number**
Rock You	Game	SQL Injection	32,603,388
CSDN	Programmer	Hacking	6,428,287
Flirtife.de	Marriage	Hacking	115,589
Faithwriters	Writing	SQL Injection	9,709
MySpace	Social	Phishing Attack	41,545

B. Model Architecture

In this work, we use recurrent neural networks which are a rich class of dynamic systems have been used for generate text in character level [1]. Recurrent neural networks can persists information entered into the system previously and then use it to predict or generate information. Because recurrent neural network can generate previously unseen informations (such as characters in password guessing), we will get some new passwords that don't appear in the training dataset. As shown in the next section, these new passwords will increase the guessing accuracy of dictionary attacks. We experimented with two different recurrent models in Section IV, one of which is Long Short-term Memory[2] and the other is Gated Recurrent Unit [3][4].

The password guessing model are organized into three major layers. The first layer is the embedding layer which is mainly used to encode the input characters into a vector. The next layer is the recurrent neural network, which is used to learn the hidden feature information in the password. At the same time, the encoded group attribute vector will be sent into this layer which is used to generate the passwords for specific user category. In the last layer, we use the fully connected layer and softmax activation function to output the probability distribution of the next character. Formally, the structure of the model can be expressed as follows.

$$\begin{aligned} \boldsymbol{e}_t &= \boldsymbol{f}_{\text{emb}}(\boldsymbol{x}_t) \\ \boldsymbol{h}_t &= \boldsymbol{f}_{\text{rnn}}(\boldsymbol{e}_t, \boldsymbol{h}_{t-1}) \\ \boldsymbol{o}_t &= \boldsymbol{f}_{\text{linear}}(\boldsymbol{h}_t, \boldsymbol{g}_c) \\ \boldsymbol{P}(\mathcal{A}) &= \text{softmax}(\boldsymbol{o}_t) \end{aligned}$$

where $\boldsymbol{x}_t$ denotes the input character at time step t, $\boldsymbol{h}_t$ denotes the hidden state of recurrent neural network, and $\boldsymbol{f}(\cdot)$ is the specific neural network function. $\boldsymbol{g}_c$ is the coding of group c. $\mathcal{A}$ denotes the alphabet has been used in password set which contains letters, digits and some special characters. $\boldsymbol{P}(\mathcal{A})$ denotes the probability distribution of each character in alphabet. The brief structure of the model at time step t is shown in Figure 1. We can use more than one recurrent layer. The number of layers is hyper parameter in our model.

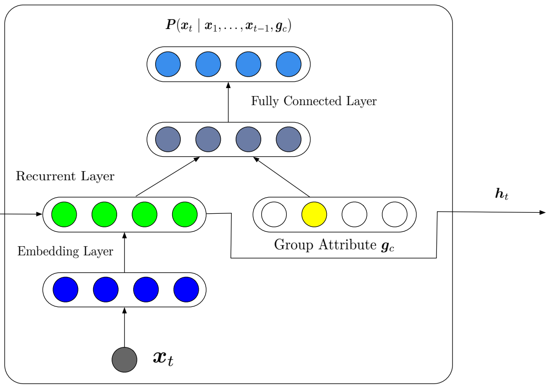

Fig. 1. The structure of the model. It accepts previous hidden state $\boldsymbol{h}_{t-1}$ and character $\boldsymbol{x}_t$ at time step t, outputs the hidden state at next moment, and the probability of the next character.

C. Model Training

To train the model, we used Tensorflow[5] , an open source deep learning library developed by Google. In the recurrent layer, there are three LSTM or GRU layers. The model was trained on two GTX1080Ti GPUs for about 18 hours. All of our code for the model was written in Python and the code can be found at github[1]. There are many hyper parameters for training the model. We experimented with many options and eventually got a better configuration. The output dimension of the embedding layer is 64 and the dimension of recurrent layers are all 256, batch size of input in the model is 32. The details of the loss function are as follows. For each training password $\boldsymbol{X}$, the loss function $L(\cdot)$ at time step t is that:

$$\begin{aligned} L(\boldsymbol{x}_t, \boldsymbol{y}_t) &= -\log\left(\frac{\exp\{\boldsymbol{o}_t^{(\boldsymbol{y}_t)}\}}{\sum_j \exp\{\boldsymbol{o}_t^{(j)}\}}\right) \\ &= -\boldsymbol{o}_t^{(\boldsymbol{y}_t)} + \log\left(\sum_j \exp\{\boldsymbol{o}_t^{(j)}\}\right) \end{aligned}$$

where $\boldsymbol{x}_t$ is the input and $\boldsymbol{y}_t$ is the label at time step t. $\boldsymbol{o}_t$ is the linear layer output. $\boldsymbol{o}_t^{(j)}$ denotes the j^{th} component in $\boldsymbol{o}_t$. The total loss $\mathcal{L}(\boldsymbol{X})$ is the sum of the loss function values for all time steps.

$$\mathcal{L}(\boldsymbol{X}) = \sum_{t=1}^{T} L(\boldsymbol{x}_t, \boldsymbol{y}_t)$$

D. Generating Passwords

E. Probability-based Password Strength Measure

The basic idea of the password strength measure is that the strength of a password is inversely proportional to the probability of its appearance. Given a group attribute $\boldsymbol{g}_c$, we can use the following equation to calculate the probability of a password $\boldsymbol{X}$:

$$\begin{aligned} \boldsymbol{P}(\boldsymbol{X} \mid \boldsymbol{g}_c) &= \boldsymbol{P}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \mid \boldsymbol{g}_c) \\ &= \boldsymbol{P}(\boldsymbol{x}_1 \mid \boldsymbol{g}_c) \prod_{t=2}^{T} \boldsymbol{P}(\boldsymbol{x}_t \mid \boldsymbol{x}_1 \ldots \boldsymbol{x}_{t-1}, \boldsymbol{g}_c) \end{aligned}$$

[1]https://github.com/luchu1993/Password-Generating-with-CharRNN

TABLE II
TABLE OF FIVE WORST/BEST PASSWORDS IN TRAINING SET

(a) Top-5 worst passwords

Password	Probability
12345678	0.002911
iloveyou	0.002900
abcd1234	0.001365
hello123	0.001355
password	0.001254

(b) Top-5 best passwords

Password	Probability
xyz2fgds3	2.450712×10^{-48}
34dfxvdc	2.565318×10^{-46}
ypp88jtj	9.192378×10^{-38}
19@96fbqaz	2.653402×10^{-36}
g8m456gz	6.811976×10^{-35}

where T denotes the length of password. $\boldsymbol{P}(\boldsymbol{x}_1 \mid \boldsymbol{g}_c)$ is a random multinomial distribution and $\boldsymbol{P}(\boldsymbol{x}_t \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{t-1}, \boldsymbol{g}_c)$ can be calculated by the model. When the probability of password $\boldsymbol{X}$ is calculated, its strength can be computed as follows.

$$\mathcal{S}(\boldsymbol{X}) = -\log \boldsymbol{P}(\boldsymbol{X} \mid \boldsymbol{g}_c)$$

where $\mathcal{S}$ denotes the password strength. Table II illustrates the five highest and lowest probability passwords in the training set.

IV. Evaluation

V. Conclusion

References

[1] A. Graves, “Generating sequences with recurrent neural networks,” *arXiv preprint arXiv:1308.0850*, 2013.
[2] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[3] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *arXiv preprint arXiv:1412.3555*, 2014.
[4] R. Jozefowicz, W. Zaremba, and I. Sutskever, “An empirical exploration of recurrent network architectures,” in *International Conference on Machine Learning*, 2015, pp. 2342–2350.
[5] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, “Tensorflow: A system for large-scale machine learning.” in *OSDI*, vol. 16, 2016, pp. 265–283.
[6] W. Melicher, B. Ur, S. M. Segreti, S. Komanduri, L. Bauer, N. Christin, and L. F. Cranor, “Fast, lean, and accurate: Modeling password guessability using neural networks.” in *USENIX Security Symposium*, 2016, pp. 175–191.