2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)

Password Guessing Based on LSTM Recurrent Neural Networks

Lingzhi Xu, Can Ge, Weidong Qiu, Zheng Huang, Jie Guo, Huijuan Lian
School of Information Security Engineering
Shanghai Jiao Tong University
Shanghai, China
qiuwd@sjtu.edu.cn

Zheng Gong
School of Computer Science
South China Normal University
Guangzhou, China
cis.gong@gmail.com

***Abstract*—Passwords are frequently used in data encryption and user authentication. Since people incline to choose meaningful words or numbers as their passwords, lots of passwords are easy to guess. This paper introduces a password guessing method based on Long Short-Term Memory recurrent neural networks. After training our LSTM neural network with 30 million passwords from leaked Rockyou dataset, the generated 3.35 billion passwords could cover 81.52% of the remaining Rockyou dataset. Compared with PCFG and Markov methods, this method shows higher coverage rate.**

Keywords-password guessing; recurrent neural network; LSTM

I. INTRODUCTION

Password guessing aims at generating password dictionaries or datasets cover as many user passwords as possible with minimum size. On one hand, these generated passwords can detect security risk of user passwords. On the other hand, these generated passwords can promote the performance of dictionary attack.

In recent years, the study of password generation methods concentrates on: (1) using better templates (e.g, probabilistic context-free grammar (PCFG) [1,5,7,13,14]); (2) using interrelationships of characters (e.g, Markov chain method [2,8,10]).

Compared with Markov-based models, which only consider relationships among several neighboring characters, recurrent neural networks (RNNs) have the ability to reveal interrelationships among all characters in passwords. Therefore, RNNs have the theoretical feasibility to improve password guessing.

Our password guessing model is based on Long Short-Term Memory (LSTM) RNNs. We train our neural network with leaked password, and generates passwords with a search algorithm. The contribution of this paper lies in implementations of password guessing based on LSTM RNNs, we also make comparisons of different number of layers/neurons. Compared with template-based method and Markov-based method, experiment results show that our method usually reaches higher coverage rate of test passwords while generating the same size passwords.

This paper is organized as follows. Section II gives a brief overview of related works. Section III describes LSTM layers and softmax regression. Section IV gives our implementation. Section V shows experiment results and comparisons in detail. Our conclusion will be given in section VI.

II. RELATED WORKS

After Weir et al. proposed PCFG which generates templates from training passwords and pads these templates with dictionaries for password guessing [1], Chou et al. developed a TDT model to generate more accurate templates and dictionaries [7]; Li et al. improved PCFG for Chinese Web passwords by adding Chinese pinyin and rules [5]; Veras et al. built PCFG with semantic and syntactic tags [14]; Houshmand et al. optimized guessing efficiency with keyboard patterns and multiword patterns [13].

Besides, Narayanan and Shmatikov first introduced Markov model in dictionary attack [10]. Castelluccia et al. estimated password strengths based on Markov model [2]. In 2015, Dürmuth et al. implemented OMEN, a password cracker based on Markov model, and guessed more than 80% of 2.6 million Rockyou passwords by generating 10 billion passwords [8].

Template-based model and Markov-based model have been applied to password cracking such as John the Ripper [11] and Hashcat [3]. These two models were also used in targeted online password guessing [12]. Other related studies focused on combination [4] and evaluation [4, 6] of these two models. According to [4], Markov-based models show better results than template-based models.

In 2016, Melicher et al. proposed that neural networks could model human-created passwords and they used neural networks in password strength measure [15]. They trained data with PGS and PGS+ dataset, and tested them with MTurk and 000webhost datasets.

III. PRELIMINARIES

A. LSTM Recurrent Neural Network

Recurrent neural networks are able to learn from sequences. To avoid gradient vanishing problem, neurons in recurrent neural network could be replaced by Long Short-Term Memory (LSTM) neurons [9].

LSTM neurons are organized into layers. The external input of LSTM layers is a matrix of data vectors. The row length of the matrix represents length of sequence, equals to the number of time steps. The column length of the matrix represents the number of sequences (batch size) in batch

978-1-5386-3221-5/17 $31.00 © 2017 IEEE
DOI 10.1109/CSE-EUC.2017.155

processing. At each time step, the external input of the LSTM layer is a column vector of the matrix.

Figure 1 is a n-neuron *m*-layer LSTM structure. At each time step *t*, the external input of the *i*-th layer is the output vector of the *(i-1)*-th layer (solid lines). The output vector of a layer will feed back to neurons of the same layer at next time step (dotted lines).

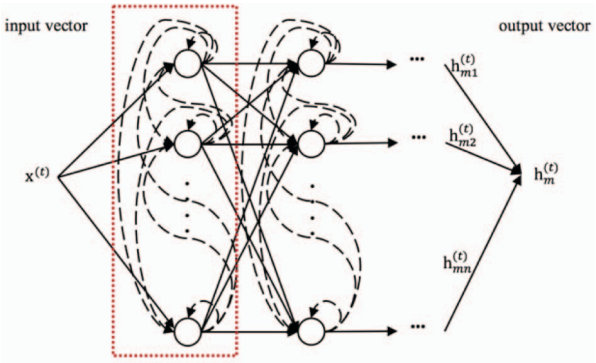

Figure 1. A Structure of LSTM Layers.

Figure 2 is the structure of *m*-layer LSTM extended by time steps. At time step *t*, the external input is the *t*-th column vector of the matrix. LSTM layers receive $x^{(t)}$ together with the last output vectors $h_1^{(t-1)}$, $h_2^{(t-1)}$, ..., $h_m^{(t-1)}$ to calculate current output vector $h_m^{(t)}$.

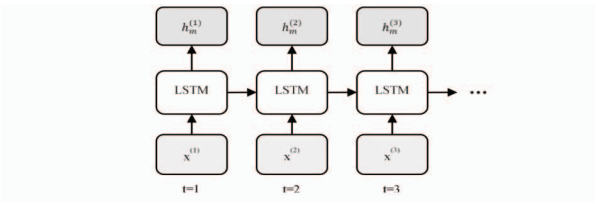

Figure 2. The extended structure of LSTM layers by time steps.

B. Softmax Regression

Softmax regression is designed for multi-classification problems [16]. The labels (categories) compose a subset *C* and the number of labels is |*C*|. Classification probability can be estimated and the hypothesis is computed as equation (1), x represents the input vector, y represents the label and θ represents the parameter of softmax regression.

$$h_\theta(x) = \begin{bmatrix} p(y=c_1|x;\theta) \\ p(y=c_2|x;\theta) \\ \vdots \\ p(y=c_{|C|}|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x}} \begin{bmatrix} e^{\theta_1^T x} \\ e^{\theta_2^T x} \\ \vdots \\ e^{\theta_{|C|}^T x} \end{bmatrix},$$

$$\theta = \begin{bmatrix} -\theta_1^T - \\ -\theta_2^T - \\ \vdots \\ -\theta_{|C|}^T - \end{bmatrix}. \quad (1)$$

IV. Implementation

A. Password Guessing Model

Our neural network includes an input layer, several LSTM layers, a softmax layer and an output layer. As it shown in Figure 3, the number of LSTM layers is marked as *m*. δ represents parameters in all LSTM layers. θ represents the parameters in softmax layer.

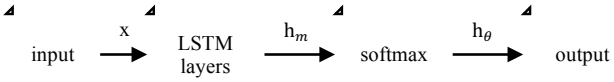

Figure 3. The Structure of our neural network.

The basic assumptions of our password guessing model include: (1) each len-length password "p1p2...plen" is a sequence x(1)=p1, x(2)=p2, ..., x(len)=plen; (2) the value of x(t) (2<=t<=len) is relevant to the sequence prefix x(1), x(2), ..., x(t-1).

The basic ideas of our password guessing model include: (1) the probability distribution of x(t) can be predicted by our neural network when using x(1), x(2), ...,x(t-1) as sequence inputs; (2) our neural network is trained by leaked passwords; (3) once given a prefix after training, the next character can be decided by a selection algorithm according to probability distribution; (4) each character (except the first character) in a password can be predicted sequentially according to (3).

B. Password Preprocessing

Leaked passwords are used to train our neural network. Leaked passwords are divided into 3 parts for our model: a training set, a validation set and a test set. The training set is applied to train the parameters in LSTM; The validation set is applied to ensure parameters trained adequately (especially prevent overfitting); The test set is used to evaluate the performance of our model.

Before using passwords for training, the preprocessing includes: (1) data cleansing; (2) data format transformation.

According to the requirement of generation, the character set and the range of password length is decided. The character subset is $C=\{c_1,c_2,c_3...,c_{ed}\}$($c_{ed}$=ED, ED represents the end of a password). Data cleansing contains three steps: (1) removing passwords which contain undesirable characters; (2) removing passwords which exceed limit length and (3) shuffling passwords.

Data format transformation aims at transforming passwords for supervised neural network. Data format transformation contains two steps: (1) character substitution (replacing characters with the label $L(c_i) = i$, $c_i \in C$); (2) sequence transformation (converting "$L(p_1)L(p_2)...L(p_{len})$" into $x^{(1)}=L(p_1)$, $x^{(2)}=L(p_2)$, ...,$x^{(len)}=L(p_{len})$,$x^{(len+1)}=L(ED)$).

C. Training Password

We train our model to approach the real probability distribution of each character in passwords.

For example, given a *len*-length training data $x^{(1)}=L(p_1)$, $x^{(2)}=L(p_2)$, ... , $x^{(len)}=L(p_{len})$,$x^{(len+1)}=L(ED)$, the supervised learning label is a sequence $y^{(1)}=x^{(2)}$, $y^{(2)}=x^{(3)}$, ..., $y^{(len)}=x^{(len+1)}$.

At each time step t (1<=t<=len), LSTM layers extract features of sequence inputs $x^{(1)}$, $x^{(2)}$, ..., $x^{(t)}$ and the m-th LSTM layer outputs $h_m^{(t)}$. The softmax layer receives $h_m^{(t)}$ and calculate the probability distribution. Compared with the supervised label $y^{(t)}$, the error can be computed [16]. For each training password, the cost function of softmax regression as follows.

$$J(\theta, \delta_1, \delta_2, \dots, \delta_m) = -1/len \left[\sum_{i=1}^{len} \sum_{j=1}^{|C|} 1\{y^{(i)} = j\} \log \frac{e^{\theta_j^T h_m^{(i)}}}{\sum_{r=1}^{|C|} e^{\theta_r^T h_m^{(i)}}} \right]. \quad (2)$$

$1\{\cdot\}$ is an indicative function, θ is the parameter of softmax layer and δ is the parameter of LSTM layers. $|C|$ represents the number of characters in subset C.

Since θ and δ represent the entire parameters of our model, the cost function calculates errors and θ and δ will be adjusted by using Back Propagation Through Time (BPTT).

D. Generating Password

This method (Algorithm 1) avoids generating repeated passwords. Since we are able to predict the probability distributions of characters, the appearing possibility of each password prefix can be estimated by probability multiplication formula.

Algorithm 1. Password Generation Method

charset: character subset $C = \{c_1, c_2, \dots, c_{|C|} = \text{ED}\}$
first_character_probabilities: statistics results of first characters
prefixes: password prefixes array
LUT: a look-up table of {key=prefix, value=probability}
threshold: minimum probability

```
//load first character probabilities
//Label function L(ci) = i, ci∈C
for c in charset:
  if (first_character_probabilities(L(c))>threshold):
    prefixes.push(c)
    LUT[c] = first_character_probabilities(L(c))
endfor;
//iterate all possible passwords' probability more than threshold
while (prefixes not empty):
  current_prefix = prefixes.pop()
  //get next possible characters' probabilities by NN
  next_char_prob = NeuralNetwork(current_prefix)
  for c in charset:
    if (LUT[current_prefix] * next_char_prob(L(c)) > threshold):
      if (c == ED): print current_prefix
    else:
      prefixes.push(current_prefix+c)
    LUT[current_prefix+c]= LUT[current_prefix]*next_char_prob(L(c))
  endfor;
endwhile;
```

E. Testing Password

The coverage rate α of test set reflects the accuracy of password guessing. The formula is shown in equation (3), n_{test} is the size of test set and n_{hit} is the size of intersection set between test set and generated passwords. The generation effect is better if α is higher while generating the same size of passwords, or α is the same while generating less passwords.

$$\alpha = \frac{n_{hit}}{n_{test}} \times 100\%. \quad (3)$$

V. Experimental Results

A. Key Parameters of RNN

Our experiments based on Theano using GTX970. We designed two experiments in order to find better key parameters: 1) compare α curves of different number of LSTM layers; 2) compare α curves of different number of neurons per LSTM layer. Finally, we generated a series of password datasets in different sizes using different parameters.

1) Hidden Layers Number (HLN): In this part, three implementations with different number of LSTM hidden layers (HLN) are implemented: one, two and three hidden layers. The number of neurons per LSTM layer is 256. Experiments were done in small sizes dataset generation, threshold are 10^{-6}, 10^{-7}, 10^{-8}, 10^{-9}. Figure 4 is a logarithmic graph, x axis stands for number of generated in logarithmic, y axis stands for α. With the decrease of threshold, the number of generated passwords increases. Figure 4 shows that the increase of HLN brings slight, unobvious increase (only about 1% at the same threshold) on generation results.

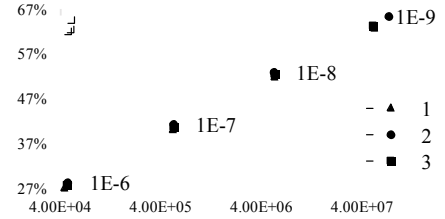

Figure 4. Different number of LSTM layers.

2) Neurons Number per Layer (NNPL): In this part, five implementations are implemented: number of neurons per layer (NNPL) are 32, 64, 128, 256, 512. All RNN implementations contains two LSTM hidden layers. Experiments were done in small sizes dataset generation but enough to identify better parameters: 10^{-6}, 10^{-7}, 10^{-8}, 10^{-9}. Figure 5 is also a logarithmic graph, x axis represents number of generated in logarithmic, y axis represents α. With the increase of NNPL, generation effects increase. Table I. shows the coverage rate increase of NNPL, and two points can be seen: (1) with the increase of NNPL, coverage rate α increases, especially when NNPL is small; (2) with the increase of NNPL, the increase of coverage rate gradually reduces. When NNPL achieves 256, coverage rate has no significant increase.

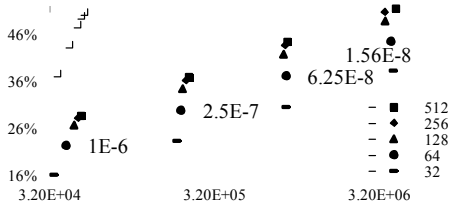

Figure 5. Different neurons per layer.

With the increase of HLN and NNPL, coverage rate increases at the same threshold, but the training and generation speed decreases. Finally, we choose a RNN with 2 hidden LSTM layers and 256 neurons per LSTM in following experiments.

TABLE I. COVERAGE RATE INCREASE BETWEEN DIFFERENT NNPLS

	1E-6	2.5E-7	6.25E-8	1.56E-8
32→64	6.12%	6.40%	6.40%	6.19%
64→128	4.18%	4.53%	4.49%	4.16%
128→256	1.54%	1.79%	1.92%	1.90%
256→512	0.45%	0.64%	0.67%	0.74%

B. Comparison with Previous Research

We sampled some large-size datasets and compared the results with previous researches. Our LSTM model contains 2 hidden LSTM layers, 256 neurons per LSTM layer.

The LSTM model is trained by 30.0 million Rockyou (RY) passwords, including 27.0 million training set and 3.0 million validation set. Besides the Rockyou test set (2.6 million passwords), Myspace dataset (MS) and Facebook dataset (FB) are also used as test sets.

We compared our implementation with OMEN-4Gram, probabilistic context-free grammar (PCFG), John the Ripper (JtR) Markov mode and JtR incremental mode.

TABLE II. COVERAGE RATE OF DIFFERENT ALGORITHMS

	Algorithm	Train Set	#guess	Test Set		
				RY-2.6M	*MS*	*FB*
1	Omen-4Gram [8]	RY-30M	1.0E+10	80.40%	77.06%	66.75%
2	PCFG [8]	RY-30M	1.0E+09	32.63%	51.25%	36.4%
3	JtR-Markov [8]	RY-30M	1.0E+10	64%	53.19%	61%
4	JtR-inc [8]	RY-30M	1.0E+10	54%	25.17%	14.8%
5	LSTM	RY-30M	3.4E+09	81.52%	77.98%	59.71%
6	LSTM	RY-30M	9.8E+08	73.77%	69.30%	50.35%

Table II. shows the coverage rate of different algorithms. Comparing 5 with 1, 3 and 4, for Rockyou test set and MySpace dataset, we hit more passwords than Omen-4Gram, JtR Markov mode and JtR incremental mode in less guesses; but the guess effect of Facebook dataset seems worse. Comparing 6 with 2, our coverage rate is much higher than PCFG in testing RY-e, Myspace and Facebook dataset.

VI. CONCLUSION

LSTM recurrent neural networks can apply to password guessing. The experiment results show our neural network is better than template-based and Markov-based model, for our generated dataset hit more passwords in most test sets.

In order to get better results, the structure of neural network may be further adjusted. Future works may focus on the difference among different datasets, study common or specific neural networks.

ACKNOWLEDGMENT

Weidong Qiu is supported by the New Century Excellent Talents in University of Ministry of Education under Grant NCET-12-0358 and the Program of Shanghai Technology Research Leader under Grant 16XD1424400.

Zheng Gong is supported by the National Natural Sciences Foundation of China under Grant No. 61572028, and the Project of Science and Technology of Guangdong (2016B010125002), and the Natural Science Foundation of Guangdong (No.2014A030313439).

REFERENCES

[1] M. Weir, S. Aggarwal, B. d. Medeiros and B. Glodek, "Password Cracking Using Probabilistic Context-Free Grammars," in *30th IEEE Symposium on Security and Privacy*, 2009, pp. 391-405.

[2] C. Castelluccia, M. Dürmuth and D. Perito, "Adaptive Password-Strength Meters from Markov Models," in NDSS, 2012.

[3] J. Steube, "hashcat - advanced password recovery," [Online]. Available: www.hashcat.net.

[4] J. Ma, W. Yang, M. Luo and N. Li, "A study of probabilistic password models," in 2014 IEEE Symposium on Security and Privacy, 2014, pp. 689-704.

[5] Z. Li, W. Han and W. Xu, "A large-scale empirical analysis of chinese web passwords," in 23rd USENIX Security Symposium, 2014, pp. 559-574.

[6] B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri and R. Shay, "Measuring real-world accuracies and biases in modeling password guessability," in 24th USENIX Security Symposium, 2015, pp. 463-481.

[7] H. C. Chou, H. C. Lee, H. J. Yu, F. P. Lai, K. H. Huang and C. W. Hsueh, "Password cracking based on learned patterns from disclosed passwords," International Journal of Innovative Computing, Information and Control, vol. 9, Feb. 2013, pp. 821-839.

[8] M. Dürmuth, F. Angelstorf, C. Castelluccia, D. Perito and A. Chaabane, "OMEN: Faster password guessing using an ordered markov enumerator," in International Symposium on Engineering Secure Software and Systems, 2015, pp. 119-132.

[9] Z. C. Lipton, J. Berkowitz and C. Elkan, "A critical review of recurrent neural networks for sequence learning,". Available: arxiv.org/pdf/1506.00019.pdf.

[10] A. Narayanan and V. Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff," in Proceedings of the 12th ACM conference on Computer and communications security, 2005, pp. 364-372.

[11] S. Designer, "John the Ripper password cracker," [Online]. Available: www.openwall.com/john.

[12] D. Wang, Z. Zhang, P. Wang, J. Yan and X. Huang, "Targeted online password guessing: An underestimated threat," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 1242-1254.

[13] S. Houshmand, S. Aggarwal and R. Flood, "Next gen PCFG password cracking," IEEE Transactions on Information Forensics and Security, vol 10, Aug. 2015, pp. 1776-1791.

[14] R. Veras, C. Collins and J. Thorpe, "On Semantic Patterns of Passwords and their Security Impact," in NDSS, 2014.

[15] 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 network," in Proceedings of USENIX Security, 2016.

[16] Stanford UFLDL. "Softmax Regression" [Online]. Available: http://ufldl.stanford.edu/wiki/index.php/Softmax_Regression.