Cracking Password Vaults with Recurrent Neural Networks

Yubo Luo Dong Yeop Lee

I. Introduction

This paper investigates how to apply neural networks to the security of password vaults. We use recurrent neural networks (RNNs) to distinguish decoy vaults from the real ones based on the prediction cost of candidate vaults in the neural model. We use three models as our classifier, model trained on decoys, model trained on real passwords and the mixed version. The results show that neural networks can provide comparable performance compared to the KL Divergence which is the state of the art cracking method on decoy vaults.

Modern password managers like LastPass provide online services which offer users the convenience of retrieving any of the passwords to their accounts by remembering a single master password. This online storage makes password vaults a popular target for attackers. Thus, strengthening the security of password vaults is a practical and important issue. Previous works, both in defense and attack focused research, have used password models that proved effective in password guessing, such as Markov and probabilistic context free grammar (PCFG). Recently, RNNs have been shown to be effective in password guessing, surpassing performance of both Markov and PCFG, but have not yet been applied to password vaults. Motivated by this fact, we proposed to apply RNNs to the security of password vaults.

There are several previous works related to this topic. Kamouflage [4] first introduced the concept of decoy vaults, which are included along with the real vault in order to force the attacker to check the passwords in the vault after a successful decryption. Before Kamouflage, attackers could easily recognize the real vault after a brute force attack, because decrypting the vault with a wrong master password would output junk information. However, Kamouflage generates decoys based on the pattern of real vault which gives the attacker a clue to accelerate their offline decryption. Then, Chatterjee et al. [5] introduced honey encryption to decoy vaults generation, in which decoys are generated on the fly, eliminating the need for storing extra vaults while still forcing the attacker to perform as much offline computation as an attack against traditional password vaults. However, Golla et al. [7] found a way to defeat NoCrack by applying KL Divergence to distinguish the real vault from decoys. Our research question here is to investigate if RNNs can have comparable or even better performance than KL divergence.

Our approach is a straightforward application of neural networks. We use different sources of password dataset to train the neural networks and then use the trained model to evaluate the similarity between the candidate password and the dataset. In neural network-based text generation, the model predicts the next possible character by giving a probability distribution over the whole vocabulary, and then usually the top-1 character is chosen. In our evaluation, we mimic this generation process. If the candidate password is similar with the dataset used in training, characters of the candidate password should have a high probability in the prediction and thus have a lower loss. The average loss of all characters of the candidate password will be the final loss for this candidate.

For the model training, we train three different models and investigate the difference of their performance. The first model is trained on decoy vaults dataset, so in this case lower loss means this candidate is more likely a decoy vault. The second model is trained on real password dataset, and here lower loss means this candidate is more likely a real vault. The third model is to combine the previous two models, and we use the average rank of the first two models as the metric.

Our contribution is twofold. First, considering that neural networks is the state of the art language model for password guessing, we apply neural networks to the security of password manager. Second, our experiments show that RNNs have comparable performance in distinguishing decoy vaults with KL divergence.

II. Related Works

In this section, we introduce some background and related literature on password vaults.

Password attacks on password vault consists of an offline phase where the attacker uses trial-decryption and an online verification step where the passwords obtained in the first step are used to test whether the passwords are decoy passwords or real passwords. In our research, we will only focus on the first step of the attack.

A. Kamouflage

Kamouflage [4] is the first work that proposed to use decoy generation to protect password vaults. Kamouflage first generates a large number of decoy master passwords and password vault pairs before encryption (10^7 is a recommended number for medium security). After generating the decoy master password and vaults, Kamouflage uses traditional encryption to encrypt all of these decoy pairs and the real pair into a single encrypted vault. When an attacker attempts to use one of the pre-generated decoy master passwords to decrypt the vault, a decoy password vault is produced. If the attacker's guess is not one of the decoy or real master password, then an error or junk information is produced.

In Kamouflage, the security level is proportional to the number of decoy pairs. The more pre-generated decoy pairs, the harder it is to distinguish the real one from the decoys. However, increasing the number of decoys increases the storage requirement. The decoy generation process of Kamouflage can be broken into the following stages. In the preparation stage, Kamouflage parses passwords into three parts: letter, digital and special character. Next, all of the passwords in the vault are tokenized into a category name and a number of certain length. For example, "Password88" is tokenized as L_8D_2, where L_8 means eight letters and D_2 means 2 digits. A probability derived from a dataset is then assigned to each token which will be used later in the generation process. Finally, in the generation stage, according to the pattern of the real master word and passwords in the vault, decoys are selected from the corresponding list that shares the same pattern with the real one.

The drawback of this generation approach is that the decoy and real passwords share a common pattern. Once the attacker makes a guess to produce one of the decoys, the pattern is exposed. The attacker can then use this pattern to speed up the search for the remaining master passwords. Chatterjee et al. [5] broke the scheme using the revealed pattern and showed that Kamouflage actually degrades the security compared to traditional methods.

B. Honey Encryption

Jules and Ristenpart[9] proposed a new system called Honey Encryption (HE) which adds another encoding procedure before the traditional encryption step. HE produces plausible-looking decoy plaintexts when decrypted with any wrong key. Unlike Kamouflage, HE can generate the decoys on the fly, reducing the space complexity. The key challenge of HE is the extra encoding part called the distribution transforming encoder (DTE). Different DTE should be created differently for different datasets. For some data, like credit card numbers, it is relatively easier to create a DTE. But for passwords created by people, building the DTE is a challenge.

Another advantage of HE is that the security strength of password vaults built on HE will never degrade below the level of traditional vaults. Honey Encryption provides security beyond the traditional brute force work bound, in contrast to Kamouflage's solution.

C. NoCrack

Chatterjee et al. [5] introduced the conception of Honey Encryption in password vaults to make them crack-resistant. NoCrack used a natural language encoder (NLE) based on PCFG[11] and Markov model separately as the DTE for the honey encryption. This enables the encrypted vaults to generate plausible-looking decoys on the fly when decrypted with any wrong master password, while Kamouflage only produces decoy vaults for pre-generated decoy master passwords.

The framework of NoCrack is shown in Figure 1. It first uses the natural language DTE to encode a plain password into a bit string and then uses the traditional password based encryption method to encrypt the bit string in to ciphertext.

However, Golla et al. [7] found a way to crack NoCrack by using Kullback-Leibler Divergence (KL divergence) to distinguish real vault from the decoy vaults. KL divergence is measure of the difference of two probability distributions. Their approach uses NoCrack to generate a great amount of decoy vaults based on which the probability distribution of decoys (P_{decoy}) is sampled. In the cracking stage, P_{decoy} is used to compare with candidate vaults. The smaller the distance between P_{decoy} and P_{cv} is, the more likely this

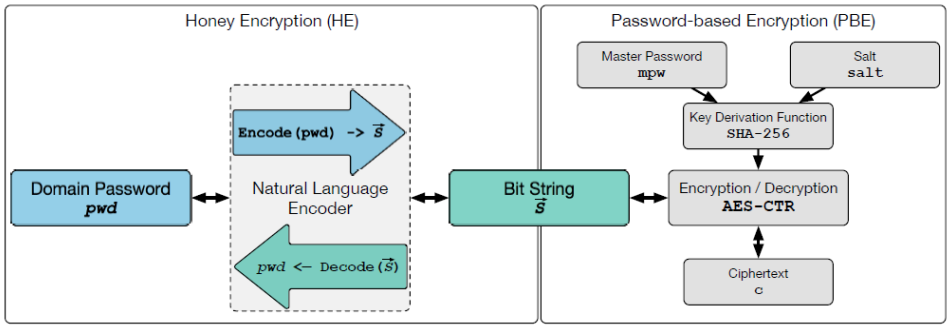

Fig. 1. The Framework of Nocrack[7]

candidate vault is a decoy. Golla et al. [7] proposed the idea of adaptive NLE, where the generated distribution of decoy vaults depends on the actual passwords stored in the vault, to improve the resistance of encrypted vaults against KL divergence attack.

III. Our Approach

There are many different language models used in password guessing. Most of them have already been investigated in decoy vaults generation, like the Markov and PCFG. Our approach is to apply language models trained from neural networks to decoy vaults generation and attacks.

There are many options available in neural network models. We choose the Long Short-term Memory networks (LSTMs), a modified version of RNNs. LSTMs have proven effective in dealing with sequential information, like text generation. Melicher [10] applied LSTMs in password guessing and achieved state of the art guessing ability. Thus, we believe LSTMs should work well in decoy vaults generation or detection.

A. LSTM Networks

The simple RNNs have the vanishing gradient problem and therefore are not good at dealing with long-term dependency. LSTMs are explicitly designed to avoid the long-term dependency problem. All recurrent neural networks have the form of a chain of repeating modules of neural networks, as shown in Figure 2.

LSTMs use carefully regulated gate structure to achieve the ability to remove or add information to the cell state. Gates are a way to optionally let information through, or forget information. We do not talk further about LSTMs details as they are out of our scope. The main takeaway is that LSTMs are good at dealing with long sequential information.

As shown in Figure 2, before we train the model, we need to first preprocess the dataset. For every password in the training dataset, we need to pad a start symbol at the beginning and a end-of-sentence symbol at the end. The networks then learns to predict the next character based on the input sequence and the prediction loss will guide the networks to optimize the neural parameters.

B. Password Vault Evaluation

The attacker's goal is to find the real vault after a brute force attack is carried out where a real password vault along with decoy password vaults is generated. Our evaluation process allows an attacker to rank the vaults based on the likelihood of being the real vault.

Suppose we have 1000 password vaults in which 999 of them are decoys. The attacker needs to somehow rank these 1000 vaults and then check whether each vault is a real vault, starting with the top ranked vault to the last ranked vault.

For example, if the rank of the real vault is 10, the attacker needs to perform online verification 10 times. Since NoCrack is designed to increase the amount of online work, we use the ranking of the real vault as our primary evaluation metric.

To evaluate the password vault, we first have to consider the single password situation where there is only one password in the vault.

The evaluation step is similar to the training process. After we obtain a trained model, we need to give the model an input and then the model gives back an output. For example, we suppose the candidate password that we want to evaluate is **Boss1234**. First, we input the start symbol into the trained model and the model gives back a probability distribution over the whole vocabulary, as shown in Figure 3. We pick up the probability of letter **B** and use the *log* function to obtain a cost value for this character.

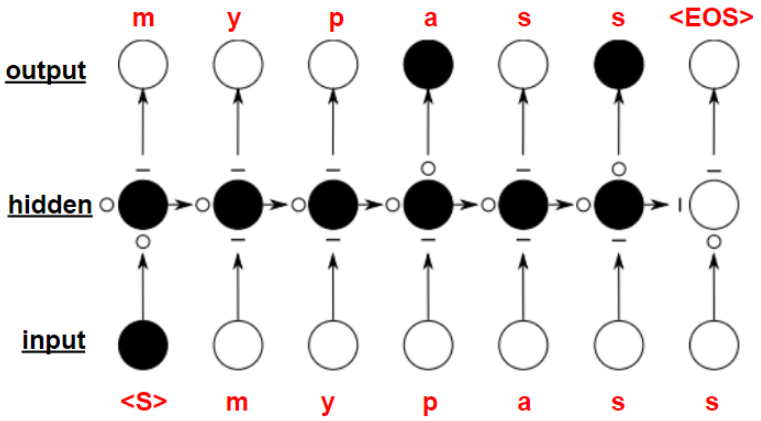

Fig. 2. RNNs Training Structure

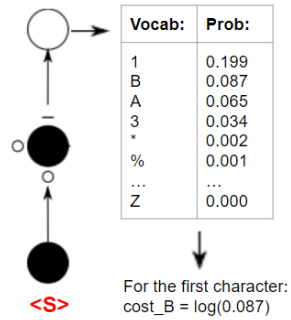

Fig. 3. Single Password Evaluation

The Candidate Vault			
Passwords	Cost	Prob	Sum
MyPass	4.334	0.4	1.7336
MyPass	4.334	0.4	3.467
Melissa	3.556	0.6	5.600
Melissa	3.556	0.6	7.334
Melissa	3.556	0.6	9.468
Final Score: 9.468			

TABLE I
EVALUATION EXAMPLE

Next, we input the start symbol and the first character **B** into the model and get back the probability distribution for the second character. We follow the same procedure and get the loss for the letter **o**. We repeat this until we finish the whole password, with the average loss as our final score for this password.

For password vaults containing multiple passwords, we use the accumulated sum of the whole vault as our evaluation score. As shown in Table I, we first evaluate every single password individually and then multiply their cost with their probability. Finally, we add all of the costs together and the total sum is our final score for this vault.

In order to better investigate our neural based evaluation approach, we use three different models. The first one is the $Model_{decoy}$ which is trained on the decoy password dataset. In this case, lower cost means that this vault is more likely a decoy vault. The second model is $Model_{real}$ which is trained on a real password dataset. In this case, lower cost means that this vault is more likely a real vault. The third model is a mixed version. We combine the first two models' results by calculating the average ranking of these two, and we reorder the list by this average ranking.

IV. EXPERIMENTS

To train our $Model_{decoy}$, we first generated 0.5 million decoy passwords by using the same method used in NoCrack to generate decoy passwords from bitstrings. We obtained the trained model by passing the decoy passwords to the recurrent neural network as input. For our $Model_{real}$, we sampled 14 million unique passwords from the RockYou password dataset and used the same network to get the trained model of real passwords.

To test our attack, we used the Pastebin Vault dataset used by NoCrack. For each real password vault, we constructed a set of 1,000 vaults by generating 999 decoy vaults using the decoy generation used by NoCrack. We scored each vault in the set using KL divergence, decoy score, real score, and mixed score. Finally, we sorted the vaults by each of the four scores to get the ranking of the real vault for each metric.

We split the vaults into three categories based on the number of passwords. The smallest category was

TABLE II
VAULT RANKING STATISTICS

	Group 2-3			Group 4-8			Group 9-50		
Metric	Mean	$Q_{0.25}$	Median	Mean	$Q_{0.25}$	Median	Mean	$Q_{0.25}$	Median
KL	106	0	12	114	0	3	36	0	0
Decoy	88	2	8	99	0	3	32	0	0
Real	170	9	40	156	2	10	64	0	3
Mixed	88	1	7	102	0	3	33	0	0

for vaults with 2-3 passwords, the next category for vaults with 4-8 passwords, and the last for vaults with 9-50 passwords.

We implemented the neural network using the Keras [6] Python library with the Tensorflow [1] backend. The RNNs used 3 layers with 320 hidden nodes and a sequence length of 20. The models were trained for a total of 10 epochs.

V. RESULTS

The results of our rankings are summarized in table II. We found that our $Model_{decoy}$ has similar performance with KL divergence, while our $Model_{real}$ performed worse than both the KL and $Model_{decoy}$. All of the ranking metrics performed best on the largest vault group, which is consistent with the results from Golla et al.

The password vaults from the Pastebin dataset might not be a representative sample of real password vaults. By manually inspecting individual password vaults in the dataset, we found that almost all vaults are human generated passwords. Additionally, most vaults reused the same few passwords with either partial or full reuse. It is not clear whether this usage pattern of passwords holds with modern password managers that try to encourage users to use randomly generated passwords. Our results demonstrate that if most passwords in vaults are human generated, then neural networks provide an effective method of ranking the real vault.

VI. LIMITATIONS

For traditional passwords, a number of datasets are publicly available. In contrast, the only known public dataset for password vaults is the Pastebin dataset. Almost half of the password vaults in the dataset only contain a single password and could not be used in our evaluation. A larger real world dataset for password vaults would allow for better evaluation of our attack method.

We used a typical set of parameters for our neural network. Hyperparameter tuning is a process in which the various parameters of the network which need to be set before training are optimized to improve the performance of the network. Many techniques exist for automating hyperparmeter tuning [2] [3]. We believe further tuning will improve the performance of our neural network based attack.

There are other techniques that can improve neural network performance, such as L_2 regularization to avoid overfitting the model and batching normalization [8] to improve learning rates. We believe that applying these standard optimization techniques can further improve the detection of decoy password vaults.

VII. CONCLUSION

Password vaults protected with honey encryption guarantees that the attacker will need to perform as much offline computation as attacking a traditional password vault encrypted with a master password and also perform additional online work in order to check for decoy passwords. Previous works have shown that the amount of online work that the attacker needs to find the real vault is much lower than claimed in NoCrack. We demonstrate that an alternative attack method using recurrent neural networks to detect decoy vaults has comparable performance with KL divergence. We believe that with better tuning and optimization of our neural networks, our ranking metrics can improve further, making this type of attack more feasible for an attacker.

VIII. ACKNOWLEDGEMENT

Yubo mainly did the background review and reimplementation of previous works. Dong was mainly responsible for the neural network implementation and the experiments and analysis.

REFERENCES

[1] ABADI, M., BARHAM, P., CHEN, J., CHEN, Z., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., IRVING, G., ISARD, M., ET AL. Tensorflow: A system for large-scale machine learning. In *OSDI* (2016), vol. 16, pp. 265–283.

[2] BERGSTRA, J., AND BENGIO, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research 13*, Feb (2012), 281–305.

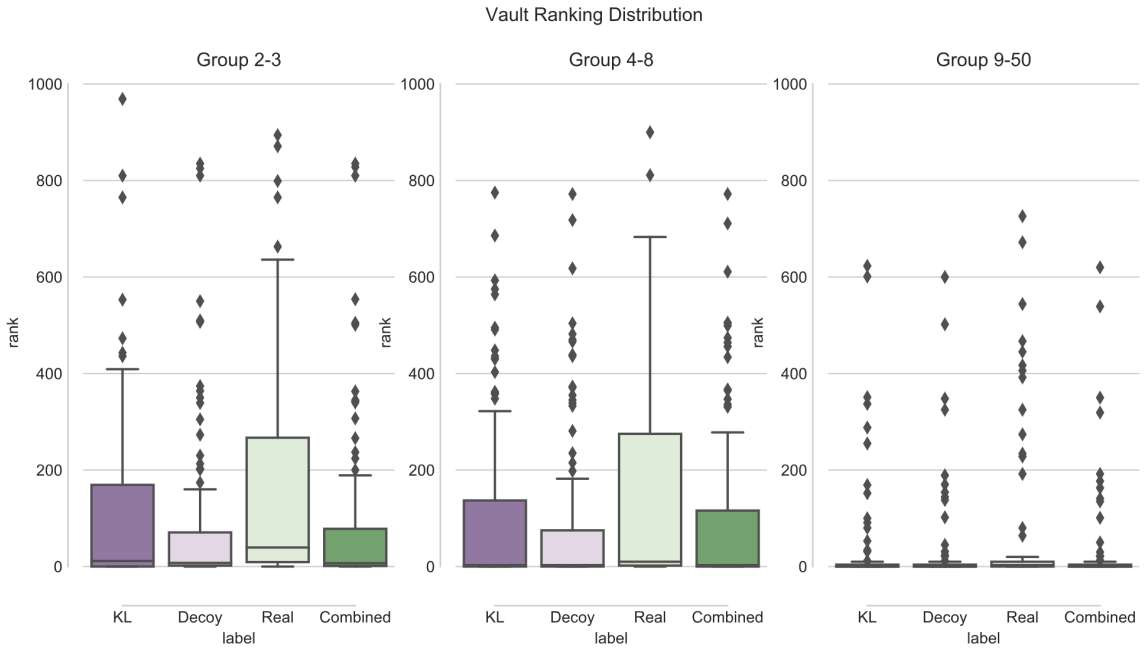

Fig. 4. Vault Ranking Distribution

[3] BERGSTRA, J. S., BARDENET, R., BENGIO, Y., AND KÉGL, B. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems* (2011), pp. 2546–2554.

[4] BOJINOV, H., BURSZTEIN, E., BOYEN, X., AND BONEH, D. Kamouflage: Loss-resistant password management. In *European symposium on research in computer security* (2010), Springer, pp. 286–302.

[5] CHATTERJEE, R., BONNEAU, J., JUELS, A., AND RISTENPART, T. Cracking-resistant password vaults using natural language encoders. In *Security and Privacy (SP), 2015 IEEE Symposium on* (2015), IEEE, pp. 481–498.

[6] CHOLLET, F., ET AL. Keras. https://keras.io, 2015.

[7] GOLLA, M., BEUSCHER, B., AND DÜRMUTH, M. On the security of cracking-resistant password vaults. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM, pp. 1230–1241.

[8] IOFFE, S., AND SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[9] JUELS, A., AND RISTENPART, T. Honey encryption: Encryption beyond the brute-force barrier. *IEEE Security & Privacy 12*, 4 (2014), 59–62.

[10] MELICHER, W., UR, B., SEGRETI, S. M., KOMANDURI, S., BAUER, L., CHRISTIN, N., AND CRANOR, L. F. Fast, lean, and accurate: Modeling password guessability using neural networks. In *USENIX Security Symposium* (2016), pp. 175–191.

[11] WEIR, M., AGGARWAL, S., DE MEDEIROS, B., AND GLODEK, B. Password cracking using probabilistic context-free grammars. In *Security and Privacy, 2009 30th IEEE Symposium on* (2009), IEEE, pp. 391–405.