Keys Your Account Goodbye: Semi-Targeted Password Cracking via Keywords

Beitong Tian†
beitong2
Computer Science, PhD
beitong2@illinois.edu

Jaron Mink†
jaronmm2
Computer Science, PhD
jaronmm2@illinois.edu

Jason Liu†
jdliu2
Computer Science, PhD
jdliu2@illinois.edu

† = co-author

ABSTRACT

We present a novel method for password guessing driven by keywords related to a particular service. By using these keywords, the model is able to make more-informed guesses, requiring fewer guesses than a completely general model. However, unlike other targeted models, this model doesn't require private or personal information such as a user's birthday or other leaked passwords. Using a set of keywords related to the service and prior known passwords from similar services, we improve the percentage of guessed passwords by up to 3% without any specific data from a particular user.

1. INTRODUCTION

Passwords have become the de facto mechanism for authentication in modern society, and as such, the ability to quickly guess a user's password poses a grave threat to that user's security. Previous works in password guessing typically involve either a general non-targeted model that is oblivious to the user base it is attacking or highly targeted models that often require voluminous sets of private information for specific users, such as their previously used passwords, birthday, etc. ([6],[10],[20],[22]).

While a general model can be freely used without any prior knowledge, it often takes more guesses to successfully find a password, because it cannot be specialized to a particular user. On the other hand, a highly targeted model may require information that is inaccessible for various reasons. We seek to create a middle-ground solution that is able to apply public information to make an educated guess, thus increasing guessing efficiency, but without requiring the use of an individual's private information. Specifically, we hypothesize that users may include keywords which are related to the service they are registering for as part of their passwords. For example, a user joining a football discussion forum is likely interested in football and may include their favorite teams or athletes as part of their password.

By somehow associating a service with some related keywords, we can have the password guessing model make more educated guesses without requiring private information.

In order to achieve this compromise, we utilize a pretrained untargeted LSTM neural network password guesser. Using readily-available leaked password sets, we utilize transfer learning to tailor the guesses to our target interest group. We then create a simplistic keyword via human intuition of the target user-base and re-sort the output of the guessing algorithm via a probability-keyword weighing function. With these two simple steps, we are able to increase the percentage of guessed passwords of our offline guessing algorithm by around 3% after 10^5 guesses from the current state of the art [14] without incorporating any user-specific information.

2. BACKGROUND

2.1 Untargeted Password Guessing Models

Untargeted password models are often trained from large leaked password datasets such as RockYou [19] and Yahoo [8]. A wide variety of password guessing models exist, which vary in how they utilize this information.

Word Mangling models such as the infamous John the Ripper [15] perform predefined text modifications on each password within the supplied word list (such as capitalizing the first letter or adding an exclamation point at the end). Due to its simplicity and effectiveness, John the Ripper remains a common tool within modern NetSec teams.

Probabilistic Context-Free Grammar models [21] assume that certain password patterns are prevalent amongst users (either due to password policy requirements or from the habitual nature of humans). Utilizing these password lists, on can automatically learn the probability of a certain base structure such as S_4D_2 (describing a password with 4 alphabetic characters followed by 2 digits) and the probability of productions such as $P(D_2 \rightarrow 12)$ (i.e., the likelihood of two sequential digits being 12). Beginning from the initial start symbol, one can sequentially apply a set of productions until a terminal string is created along with a corresponding probability. By guessing the explored terminal strings in descending probability, one can crack effectively guess passwords.

Neural Networks excel in their ability to learn complex non-linear functions via a large set of inputs and outputs. Advances in neural network architecture now allow for mapping based on both current and past input and have been successfully used in applications such as text generation. As researchers have recently discovered, this is heavily applicable to the realm of password guessing [14]. Both deep neural networks and **Markov Models** guess the next character based on some of the context of the previous characters [13]; however, deep neural network architectures such as LSTM are generally much smaller than Markov Models and have the ability to easily transfer knowledge amongst different (but similar) tasks.

Whatever the guessing algorithm, each only utilizes information gleaned from large and purposely general password

sets without consideration of the user base being attacked. While this allows for great portability of a trained model, it also ignores any user-base specific information that could help improve the guessing efficiency.

2.2 Targeted Password Guessing

Most targeted models are untargeted models that have been slightly augmented to prefer including a portion of private information of a specific user that it is guessing against. Some of these fields that prior work have considered utilizing include: **First and Last Name**, **Email**, **Education/Work**, **Location of Residence**, **Birthday**, **Cell Phone**, **Siblings**, and **Previous Passwords**.

As Sun et al. have demonstrated, utilization of this information in offline attacks can decrease guess count from 200 million to 500 thousand for the same number of correctly guessed password and can result in x3-x6 more passwords cracked in an online case where only 100 guesses for each user are allowed [10].

While obviously valuable, much of this information is traditionally considered private and thus may not be possible for an attacker to obtain. Furthermore, even if the information is potentially obtainable, a substantial amount of effort must be made by the attacker to collect each user's information, which could dramatically outweigh any potential benefits the targeted algorithm presents. Thus to be useful, the external information incorporated within a targeted attack should ideally require little effort and be publicly available.

2.3 Recurrent Neural Networks

Recurrent neural network (RNN) is a popular neural network architecture for sequence processing. Unlike the feedforward neural network such as convolutional neural network (CNN), recurrent neural network will reuse the neural cell along a time sequence. Its basic structure is a network with a loop, which means its output of the network is fed back to itself. If we unroll the recurrent neural network, we can find its architecture is like a sequence of recurrent neural network cells. Each cell learns from part of the sequence and passes the knowledge to the next cell. The sequential nature contributes to the recurrent neural network to be the most efficient model for sequence processing. Besides the vanilla recurrent neural network, researchers have created many other variants of recurrent neural network such as long short-term memory(LSTM)[9] and Gated Recurrent Units(GRU)[7] to increase the performance of RNN.

2.4 Transfer Learning

While advances in machine learning algorithms have allowed well-trained models to outperform the vast majority of human crafted models and heuristics, these trained models are only particularly adept at the very narrow task for which they are trained. The goal of transfer learning algorithms is to utilize the well-learned features for a source task and apply it to a new target task to improve performance.

Neural Network models have been shown to learn hierarchical sets of features within each layer [23]. Typically, the beginning layers learn low-level features, while the higher layers utilize these low-level features to create patterns that eventually represent complex concepts. As an example, a CNN trained on a categorizing animals will likely learn noisy, simple shapes such as lines, curves, and textures within the lower layers; conversely, the higher layers will learn how to combine these patterns to form legs, bellies, heads, and ultimately a whole concept such as a cat near the top[23]. It follows that transfer learning within Neural Networks usually involves freezing a subset of the lower layers while retraining the higher layers. Intuitively, this reflects the idea of maintaining the well trained low-level patterns that are common amongst the source and target task while retraining the high-level patterns which differ between the specific tasks.

Transfer learning is most applicable when the source task is very well trained with a large amount of examples, while the target task has a relatively low number of examples. Utilizing the source via transfer learning allows for these well-defined patterns to be used in a new context, and can lead to a significant improvement compared to solely training on the small number of examples from scratch. Additionally, transfer learning is only helpful when the two domains share underlying commonalities between each other. Furthermore, a poor fit between source and target tasks can even lead to degraded results. A principled method of determining what tasks are transferable is still an area of active research.

3. THREAT MODEL

In this work, we assume an adversary who wishes to efficiently perform an offline attack for a specific website whose hashed user passwords have been stolen. We assume that our adversary has public knowledge of the target interest group at hand from which a set of related keywords can be derived. This information can be gathered via publicly available knowledge graphs (such as YAGO[18], DBPedia [5], and Google Knowledge Graph[17]), web scraping interest related forums (such as Reddit or website-specific forums), or by web scraping the targeted website itself.

Additionally, we assume that our adversary has a list of prior leaked password sets of websites, some of which belong to a similar interest group of our target. While this requirement may appear to substantially decrease the plausibility of this attack, we argue that there is already a voluminous amount of leaked password sets, the union of which form a nearly complete set in terms of their interest relevance. For example, "Have I Been Pwned" tracks over 9 billion passwords from 417 different websites as of December 16[3]. Everything from real estate sites to religion-specific dating websites[3] has seen breaches, and thus an attacker would not have an issue finding a marginally related prior leaked password set.

With that in mind, in future work, we do intend to explore a weaker threat model that only utilizes publicly available knowledge of interest groups via web scraping and thus alleviates this assumption.

4. METHODOLOGY

Our guessing system has three main sub-models as shown in Figure 1: (1) sequence deep neural network model; (2) targeted transfer learning model; (3) targeted sorting model. In the following subsections, we will introduce those models into details.

4.1 Neural Network Model

Our system skeleton is a recurrent neural network(RNN) model. RNN is a suitable model to process sequential data, such as text streams. Among RNN models, we choose to use

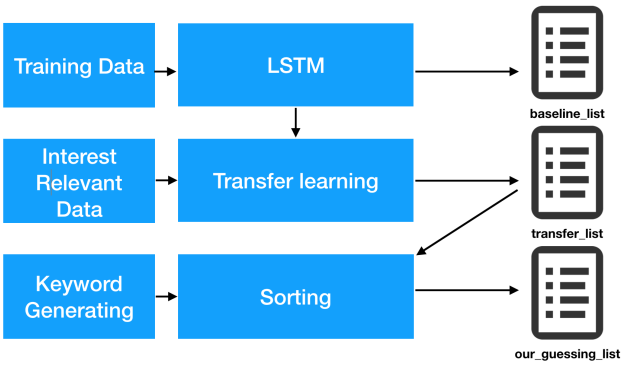


Figure 1: System diagram. Our guessing system has three main components. The first one is a pre-trained LSTM model, and we denote the guessing list generated by this model by *baseline_list*. The second part is a transfer learning model with the leaked password dataset as the transfer training data. We denote the guessing list from this model by $transfer\_list$. The last part is the sorting module. The input of this sorting module is $transfer\_list$ and *keyword_list* generated from keyword generating module and the output is our guessing list: *our_guessing_list*

a long short-term memory model(LSTM). LSTM leverages some memory gates to memorize the context of a specific piece of string, that can benefit the usage of context information for guessing. We use pairwise passwords to train the LSTM to make our model understand and learn how to compose a password. In the guessing process, we will feed a random noise to the model, and the model will generate a password with its probability. We sort the guessed passwords based on the probability and get a general guess list. The performance of the LSTM model depends on many factors, such as the layer number, the size of the hidden layer, the size of the training data, etc.

Limited by the computation source, we choose to use a CMU's pre-trained model[14]. We tried to build our LSTM. Limited by the computation source, our LSTM model has only one LSTM cell layer, 128 hidden states for each hidden layer, and we trained this model with only 100 thousand passwords. The performance of this LSTM is not satisfied. CMU pre-trained LSTM model has three LSTM cell layers, 1000 hidden states for each hidden layer and the author of this model uses 150 million general passwords to train this model. This model is proved to have a good performance on general password guessing.

4.2 Transfer Learning on Similar Datasets

The second component of our method is a transfer learning model. We want to modify the bias of passwords guessed by our guessing model. We want to freeze the weights of low layers to maintain some most basic guessing knowledge learned from the general training dataset. For example, during the general training, the model has already learned some frequent transformations for passwords such as changing *s* to $ and some most frequent sub-words used in passwords such as 123 and *qaz*, etc. After freezing the weights in the lower layer, we will retrain our model with a new targeted training dataset, such as a password list with a specific topic or a keyword list for one interesting area. Transfer learning can benefit our targeted guessing system in two folds. Firstly, it can increase the probabilities of those related passwords, which has already in the general guessing list. Secondly, transfer learning can help our model to generate more new passwords related to our targeted topic. We find the CMU password guessing system provides a transfer learning API for future learning. After comparing their implementation and our ideas, we choose to use their transfer learning API.

4.3 Sorting Algorithm

The last part of our guessing system is a sorting module. We observe the guessing process of the neural network is hard to control. We cannot easily control the influence of transfer learning on the original model.

One method is that we can try to use different hyperparameters of transfer learning and use different transfer training dataset to retrain our model and refine our guessing model. But retrain a model is resource-consuming, and the retraining process is still a black-box process.

Our method is to add one more sorting module at the last of our system. We will resort the password guessing list based on the sort weight of each password as shown in Eq 1.

$$W(pw_i) = \alpha \times Prob(pw_i) + (1-\alpha) \times Cor\_Bonus(pw_i) \quad (1)$$

In equation 1, we use the original guessing probability and correlation bonus value to represent the sort weight for each password. α is the parameter to weight the original probability and the correlation bonus value. If we want to put more weights on the original model guess results, we can set a higher α. If we want to list those passwords which are more correlated to our targeted topic, we can set a lower α.

The original guessing probability of the password means how probable our transfer learning model guesses out this password. We directly use the probability generated from our model and use Z-score function(Eq 2) to normalize the probability and get $Prob(pw_i)$ as shown in Eq 3.

$$Z(x_i) = \frac{x_i - \overline{x}}{s} \quad (2)$$

$$Prob(pw_i) = Z(Prob_{original}(pw_i)) \quad (3)$$

The correlation bonus value shows the correlation between the password and our targeted topic. We use the minimum password-keyword Levenshtein distance to represent this value. And we use min-max algorithm to normalize the result as shown in Eq 4.

$$Cor\_Bonus(pw_i) = \min_{kw_i \in keywords} (Levenshtein(pw_i, kw_i)) \quad (4)$$

The keyword list contains many relevant words and terminology commonly used within the user base community. For example, if our targeted topic is an animal shelter, the keyword list should contain many pet related terminology such as dog, cat, bearded dragon, woof, meow, and such. The generation of the keywords are described in the following subsection (4.4).

The Levenshtein distance algorithm is a classical algorithm to calculate the distance or dissimilarity between two sequences. In our model, we use this algorithm to calculate the distance between two strings: the keyword and the password. To be more precise, we want to use the Levenshtein distance algorithm to check if the password contains

the keyword or part of the keyword. During our research, we found the length difference between the keyword and the password will generate undesired results. For example, the Levenshtein distance between dog123456 and dog is 6. The Levenshtein distance between car and dog is 3. If we use the original Levenshtein distance algorithm, we will wrongly assume car is more related to our keyword dog than dog123456. In order to remove the influence of length, we add a normalize factor in the original Levenshtein distance algorithm, as shown in Eq 5.

$$Levenshtein(a,b) = Lev(a,b)/abs((Len(a) - Len(b) + 1)) \quad (5)$$

After calculating the sorting weight for each password, we will sort the guessing list based on the weight and generate our final guessing list.

4.4 Keyword Generation

In order to re-sort the guesses, a list of relevant keywords must be generated. The keywords should ideally represent a set of terminology which is used by the interest group in a higher frequency than normal parlance. Conceptually, the sorting algorithm should utilize these keywords and rearrange the guessing list to one that is consistent with the frequency of certain these keywords and phrases within the targeted password set.

For this iteration of our work, we discovered a list of keywords via human intuition of the subject at hand. In the case of Neopets (a game in which you play and raise a virtual pet), we utilized a list of consisting of the following: the word "neopets", a Wikipedia list of common animals [1], and a list of the fictitious species of animals within the game [2]. Unfortunately there are many pitfalls with such a manual method. First, it is hard to validate if the added keywords are relevant. Second, it is easy to exclude potentially useful keywords. Third, any efficiency benefits that one can realistically obtain via keyword sorting can easily be lost via manual effort.

Fortunately, there has been considerable research done on keyword discovery within text mining [11] [12] [16] . Ideally, similar algorithms can be used on the text found by web scraping the victim website or forums related to the interest group. Not only would such an algorithm allow for meaningful words to be automatically discovered via metrics like TF-IDF but one can also numerically estimate how important individual keywords are. Given individual keyword weights, one can potentially perform a sorting algorithm that is based both on the probability to keyword weighing factor α, and the keyword prevalence factor β =TF-IDF. However, this is left as future work.

4.5 Other Possible Sorting Algorithms

During the research, we also come up with some other sorting algorithm. Instead of weighting the original probability and the correlation bonus, we try to use only the correlation bonus to sort the guessing list bins. We first separate the guessing list into several bins. For example, we separate the 10^5 length guessing list into ten bins so that each bin contains 10^4 guesses. Then, we use the correlation bonus value of each password as the sorting weight to resort to each bin. This method will make sure the password in bin#1 will never fall into other bins like bin#5, which keeps the original guessing order between bins. We can change the bin size to adjust how much do we want to keep the original order. This sorting algorithm is gentler compared to the sorting algorithm we discussed in Section , and it is a more general version of the algorithm in Section if we set the bin size to 1. Based on our result, we abandoned this algorithm. We also tried to bin the guessing list in an exponential way. For example, the n_{th} bin contains 2^n guesses. The result is also not better than the algorithm in section . So, we don't consider these algorithms in later experiments.

5. EVALUATION AND DISCUSSION

5.1 Effect of Transfer Learning

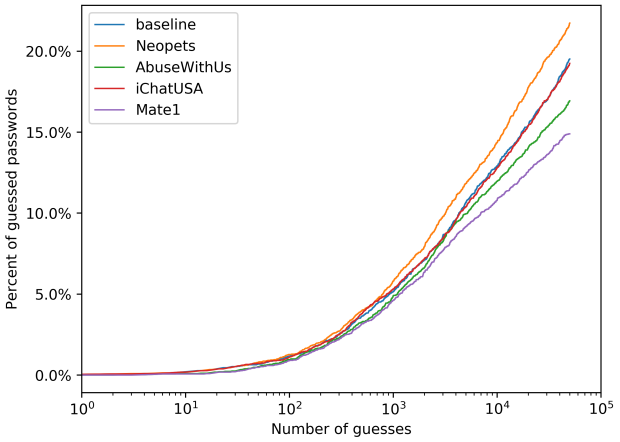
We compare the guesses generated by the model after transfer learning on various password sets. We picked datasets from three different "genres" of password sources: Neopets and Neofriends, AbuseWithUs and CrackingForum, and Mate1 and iChatUSA.

Neopets is a site featuring games and "neopets," or virtual pets. Neofriends is a discussion forum specifically about Neopets. AbuseWithUs and CrackingForum are both discussion forums about hacking or cracking accounts in general. Mate1 and iChatUSA are online dating sites.

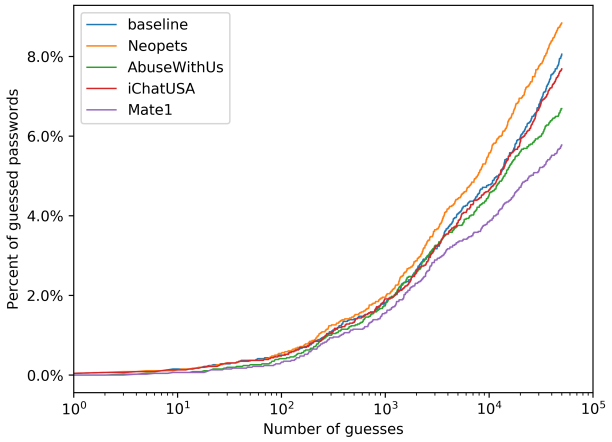
Based on our hypothesis of password similarity, we expected models trained within one genre to perform better within its own genre than across other genres. However, as seen in Figure 2, this is not what we observed. Instead, the relative performance of each model was essentially unchanged across all targets, with the Neopets model significantly outperforming all other models (including the general baseline model) on all targets. Curiously, Mate1 always performed the most poorly, even when attacking its own dataset. (The datasets were partitioned into training and testing sets, so this does not mean that Mate1 was directly attacking its own training data.)

These results raise many key questions. Why is the Neopets model so much better at guessing than any other model, and why is Mate1 so much worse? What are the fundamental differences between the password lists? While the results might suggest at first that there is no real similarity across genres, there are clearly characteristics of passwords within each set that makes them more or less favorable to the overall performance of the model. For example, on manual inspection, the dataset for Mate1 contains many email addresses, while no other dataset really contains any email addresses. We hypothesize that the transfer learning methodology we applied significantly hurt the performance of the Mate1 model – namely, freezing the features of the baseline model likely prevented the model from adjusting its features to better fit Mate1 passwords. For example, we observed in the Mate1 guesses that it never chose to guess anything email-like (not a single '@' appeared in the guesses) – despite the relative prevalence of emails in the training data. There is still significant work to do on both selections of training data and the actual transfer learning methodology for the models.

While keeping these pitfalls in mind, we do observe some semantic differences in the passwords guessed by each model. The very first guesses (within 100 or so) are typically the same across all models, containing passwords such as `password`, `princess`, `iloveyou`, and `123456`. Curiously, Mate1 is missing this last example, and instead has some less frequent numbers (such as `345678`) as well as some strange characters (such as `::::::::` or `;:987653`). There are a few colons and semicolons in the training data, but not many; most



(a) Target: Neofriends



(b) Target: mate1

Figure 2: Comparison of guesses from models trained on different datasets. Each graph shows the performance of the models when attacking a particular password set. The baseline on each graph is the general model, while each other line shows the performance of a model that underwent additional transfer learning on the specified dataset. See Appendix A for more examples.

semicolons are due to HTML encodings, such as in `ilove-dogs&me1`. This may be another feature that the model was unable to learn due to freezing that caused its degraded performance. Around 1000 guesses are when more significant divergences appear – Mate1 tends to guess names such as `madonna` or `melisa` slightly earlier than Neopets, while Neopets guesses animals such as `turkey` or `dragon1` slightly earlier. However, we do also observe that the guesses from each model do not actually differ by a great margin. Adjusting the method to guess more targeted results (and thus further differentiate each model) could lead to more interesting results. Additionally, due to restricted computing power and time, we restricted the size of the training sets for AbuseWithUs, iChatUSA, and Mate1 to a small subset of the original dataset size (∼10000 passwords). A more fair comparison may require larger training sets.

5.2 Effect of Keyword Sorting

We next consider the performance of guesses before and after sorting based on keyword similarity. Based on our general observations that around 10% of passwords have some relation to the dataset (and thus to the keywords), we choose $\alpha = 0.9$. Testing other values of α shows that the performance is not significantly affected for α around 0.9, while lower values of α (which put more weight on keyword similarity) generally reduces the effectiveness of the model, as seen in Figure 3. Looking at the order in which passwords are guessed as α changes, we see that for lower α such as$\alpha = 0.3$, the model tries too hard to guess passwords that are "related" and misses the common "easy" passwords, such as `123456789` or `password`.

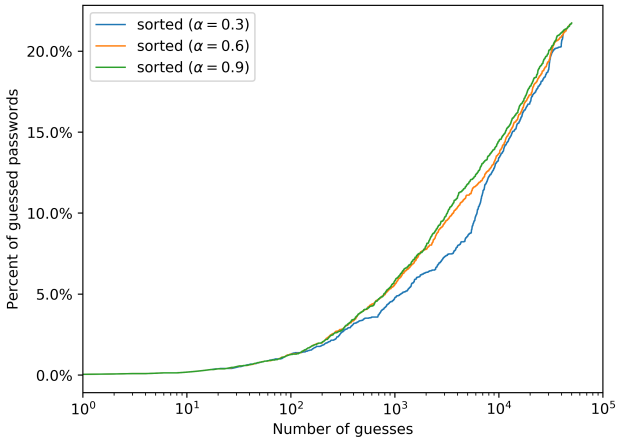


Figure 3: Comparison of guesses generated by the baseline model, Neopets transfer model, and Neopets transfer model with sorting when attacking Neofriends passwords.

Figure 4 shows the effect of sorting on the guessing performance of the Neopets model. Sorting is able to slightly boost the guessing performance as the model guesses more correct passwords slightly earlier. However, there is ultimately not a very significant gain in performance from the naïve Levenshtein sorting method. A continuation of this work would be to investigate better keyword correlation metrics. Levenshtein distance does not include any notion of semantic similarity; it only favors guesses that happen to "look like" the keywords, even if they have no real semantic similarity. For example, the guess list for Neopets with $\alpha = 0.3$ guessed `trigger` fairly early, partially because the distance between `trigger` and `tiger` is 2; the words look very similar. On the other hand, they have no real semantic similarity. Four targets in the Neofriends testing set include `tiger`: `tigers`, `tiger4505`, `whitetiger`, and `neontiger`, but `trigger` does not appear in the testing set at all.

5.3 Evaluation of Keyword Sets

The keywords in the previous sections were generated relatively arbitrarily, as aforementioned in methods. We seek to address two questions here:

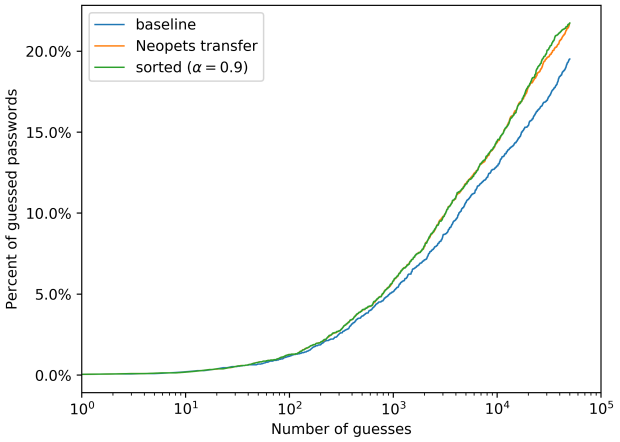


Figure 4: Comparison of guesses generated by the baseline model, Neopets transfer model, and Neopets transfer model with sorting when attacking Neofriends passwords.

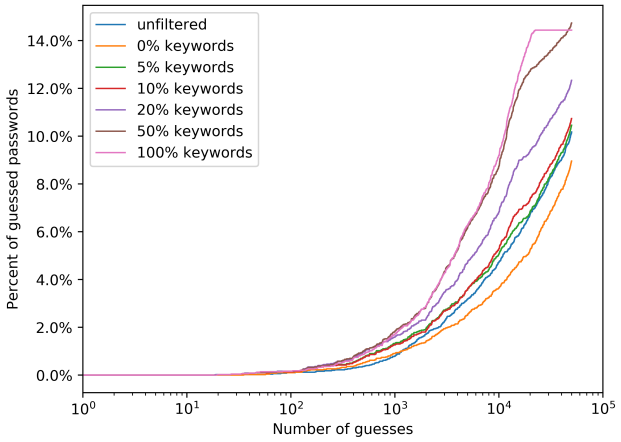


Figure 5: Comparison of guessing performance versus the keyword presence in the target set. Here, the guesses are generated from the Neopets model, and the target sets are all variations of the Neofriends password sets (adjusted to have more or fewer passwords strongly related to the keywords).

1. How well would our model perform if we had an "oracle" keyword set that perfectly encapsulates the target set in as few keywords as possible?

2. How well does our arbitrarily-selected keyword set relate to the target password set?

We address this by creating modified target sets. In each modified set, we take the testing password list and sort it according to our keyword similarity metrics. We then output target sets that contain $X\%$ passwords that are strongly-similar to the keyword set. In Figure 5, we then compare the results from attacking each modified set with our sorted guess list. As expected, the performance of the model improves as the target set becomes more similar to the keyword set. The unfiltered set is simply the target set unmodified; it is between the 0% and 5% lines, but significantly closer to 5%, suggesting that our naïve keyword list covers about 4% or so of the target set.

The general shape of these curves indicate that with a better keyword set, the performance of our password guessing model could be significantly improved; interestingly, there appear to be significantly diminishing returns between 50% keywords and 100% keywords. Part of this is also due to experimental error – there are only so many passwords in the original Neofriends set that are actually strongly related to the keywords, so the "100%" related set may struggle to find related keywords. Additionally, as mentioned before, the notion of relatedness is purely from Levenshtein distance in this paper, while we are actually seeking semantic similarity. Given a better keyword similarity metric, it may be possible to adjust the curves, so they have more of a concave-down asymptotic appearance rather than concave-up (i.e., aim to successfully guess passwords earlier). The aim is to more rapidly guess the related passwords until the model runs out of good targeted guesses and resorts back to more generic guesses.

6. RELATED WORK

AlSabah et al. observed general traits of passwords used for a Middle Eastern banking service.[4] While they classified most passwords as keyboard patterns (i.e., passwords formed by character sequences influenced by keyboard layouts), they also found that some common words appear relatively frequently in passwords, such as `qatar` ($0.5 - 1.5\%$, depending on the user's primary language) or the bank's name itself (0.6%). Names also appeared in about 25% of passwords. About 80% of passwords were a string of characters followed by some digits. Finally, they did observe notable differences in the passwords depending on a user's primary language. For example, almost 4% of Arabic users included their phone number in their password, while only about 0.06% of English users included their phone number. Although a banking service is not associated with any obvious interests, these results do show that some amount of related keywords (such as the bank's name) may still appear within the password.

Castelluccia et al. were one of the first to and exploit the relationship between user PI and password construction with a method they named "OMEN+".[6] Utilizing a 3-gram Markov model to guess the most probable next character. By assigning the transition and initial probabilities to approximate bucket levels, OMEN is able to guess passwords ordered by their approximate decreasing probability. The authors then utilized publically-available user PI by associating their registered email with their Facebook account and retrieving their first and last name, username, friends, education/work, contact info, location of residence, birthday, and siblings. By boosting PI associated character grams with their corresponding usage seen in the password set, they were able to improve their password guessing performance up to 5% at 100 million guesses.

Sun et al. developed a method they call "Personal-PCFG," which essentially uses a grammar to structure passwords, then guesses by filling in characters according to the grammar.[10] Their "Personal-PCFG" extends the typical PCFG

grammar of letters, digits, and symbols with personal information, such as a user's name, email address, cell phone number, etc. Targeting a Chinese railroad site, they observed that almost 60% of passwords contained some personal information. In terms of password structure, they also observe that Chinese passwords tend to consist mostly of digits, and are often keyboard patterns as well.

Wang et al. [20] develop a PCFG based targeted password cracking framework TARGUESS. TARGUESS embeds various personal information(PI) such as username, birthday and cell number, etc. into several tags. Unlike the length-based tag used by Li et al., TARGUESS develops a type-based tag system(tag B for Birthday, tag N for the name). The subscript variable for each category's tag restricts a specific format for the information. For example, B_3 represents Birthday in MDY format, and B_4 means Birthday in YMD format. TARGUESS can also leverage the previous password, and some categorical information like language, gender to improve the guessing performance. Wang can use detailed PI for each user to shrink his PI dictionary while we only have a general service topic and related words which can be easily obtained from public resources. We have a more challenging threat model than Wang.

7. OPEN PROBLEMS

Now we are generating keyword lists with our heuristics of the targeted topic. In future work, if we want to make our system generalized for other topics, this method is not feasible anymore. We need to develop a new method to generate keyword lists, such as using web-crawling techniques automatically. We also want to give each keyword a score to show the relevance between the keyword and the targeted topic. For example, if our guessing topic is pet, dog is more probable than chinchilla to be chosen as a part of the password. We can use this additional score to improve the accuracy of our similarity metric.

Currently, we are using the minimum Levenshtein distance algorithm to calculate the similarity between the password and the keyword. This method is effective but may not be the best distance metric. We can try to use some machine learning models to calculate the distance semantically.

There are also issues with the transfer learning method we use, as mentioned in the evaluation and discussion. Some datasets result in severely degraded performance, potentially because they contain features that are not captured by the baseline model. Because the current transfer learning method freezes the feature layers, the model is unable to properly learn from the training set, resulting in poor guesses. We still need a better method to apply the prior knowledge from other datasets (or ideally, even from a keyword list) to the password guess generation step from the model.

8. CONCLUSION

We show that there are still many possible improvements to neural network password guessing models, and that a semi-targeted model that guesses passwords based on some prior knowledge of a general service is potentially feasible, but requires additional work. There is some performance gain from training on passwords from a related dataset (e.g., from Neopets and attacking Neofriends), but there are still pitfalls to address regarding the effect of the training data on the resulting model. Furthermore, the transfer learning methods used in this study seem to have limited effectiveness because the features learned from the baseline model may not completely capture the transfer learning training set. Regardless, there is some promise in the theoretical performance of a model with an oracle keyword set that could learn perfectly even using our naïve keyword similarity metrics.

9. ACKNOWLEDGMENTS

We extensively used Melicher et al.'s open-sourced password guessing model as our baseline model and for transfer learning[14]. We also thank Gang Wang for his input and guidance during this project.

10. REFERENCES

[1] List of animal names. https://en.wikipedia.org/wiki/List_of_animal_names.

[2] Neopets wiki. https://neopets.fandom.com/wiki/Neopets_Wiki.

[3] Pwned websites. https://haveibeenpwned.com/PwnedWebsites.

[4] M. AlSabah, G. Oligeri, and R. Riley. Your culture is in your password: An analysis of a demographically-diverse password dataset. *Computers & Security*, 77:427 – 441, 2018.

[5] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, editors, *The Semantic Web*, pages 722–735, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[6] C. Castelluccia, A. Chaabane, M. Dürmuth, and D. Perito. When privacy meets security: Leveraging personal information for password cracking, 2013.

[7] 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.

[8] D. GOODIN. Hackers expose 453,000 credentials allegedly taken from yahoo service., July 12, 2012. http://arstechnica.com/security/2012/07/yahoo-service-hacked/.

[9] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] Y. Li, H. Wang, and K. Sun. A study of personal information in human-chosen passwords and its security implications. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, April 2016.

[11] Z. Li, D. Zhou, Y.-F. Juan, and J. Han. Keyword extraction for social snippets. In *Proceedings of the 19th international conference on World wide web*, pages 1143–1144. ACM, 2010.

[12] Z. Liu, X. Chen, and M. Sun. Mining the interests of chinese microbloggers via keyword extraction. *Frontiers of Computer Science*, 6(1):76–87, 2012.

[13] J. Ma, W. Yang, M. Luo, and N. Li. A study of probabilistic password models. *Proceedings - IEEE Symposium on Security and Privacy*, pages 689–704, 11 2014.

[14] 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 *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 175–191, 2016.

[15] A. PESLYAK. John the ripper, 1996-. https://www.blog.google/products/search/introducing-knowledge-graph-things-not/.

[16] S. Rose, D. Engel, N. Cramer, and W. Cowley. Automatic keyword extraction from individual documents. *Text mining: applications and theory*, 1:1–20, 2010.

[17] A. Singhal. Introducing the knowledge graph: Things, not strings, 2012. https://www.blog.google/products/search/introducing-knowledge-graph-things-not/.

[18] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 697–706, New York, NY, USA, 2007. ACM. http://doi.acm.org/10.1145/1242572.1242667.

[19] A. Vince. If your password is 123456, just make it hackme, January 20, 2010. http://www.nytimes.com/2010/01/21/technology/21password.html.

[20] 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*, pages 1242–1254. ACM, 2016.

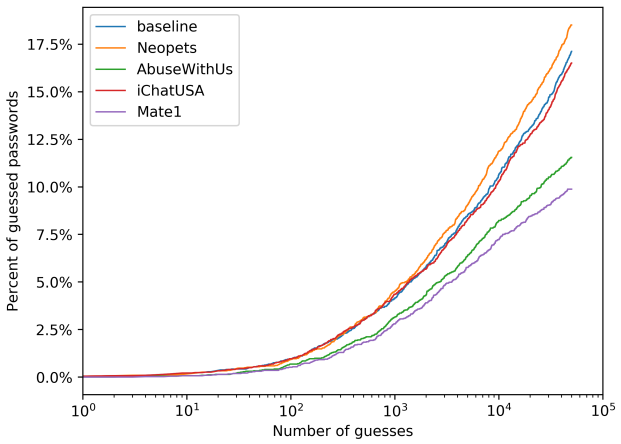
[21] M. Weir, S. Aggarwal, B. d. Medeiros, and B. Glodek. Password cracking using probabilistic context-free grammars. In *2009 30th IEEE Symposium on Security and Privacy*, pages 391–405, May 2009.

[22] M. Zhang, Q. Zhang, W. Liu, X. Hu, and J. Wei. A systematic targeted password attacking model. ksii transactions on internet and information systems. In *KSII Transactions on Internet and Information Systems, vol. 13, no. 5*, pages 2674–2697, 2019.

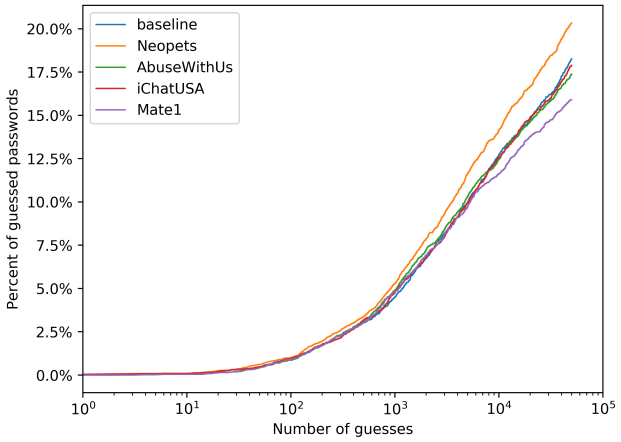
[23] Q. Zhang, R. Cao, F. Shi, Y. N. Wu, and S.-C. Zhu. Interpreting cnn knowledge via an explanatory graph, 2017.

APPENDIX

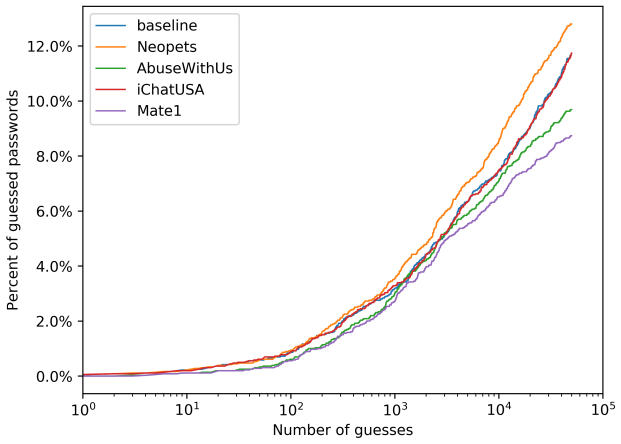
A. ADDITIONAL RESULTS



(a) Target: CrackingForum



(b) Target: AbuseWithUs



(c) Target: iChatUSA

Figure 6: Comparison of guesses from models trained on different datasets. Each graph shows the performance of the models when attacking a particular password set. The baseline on each graph is the general model, while each other line shows the performance of a model that underwent additional transfer learning on the specified dataset.