CSCE439 Final Project Research Paper

Brown
Texas A&M student
Department of Computer Science
College Station, USA

Margam
Texas A&M student
Department of Computer Science
College Station, USA

***Abstract*—The robustness of password authentication systems against guessing attacks is very important. This research aims to reproduce the findings of the work ”Modeling Password Guessing with Neural Networks” by Leo de Castro et al. Our study utilizes Long Short-Term Memory (LSTM) neural networks to model the probability distribution of passwords and a Monte Carlo technique for guess number estimation, thereby estimating the resistance of passwords to guessing attacks and evaluating password strengths. Our work validates the efficacy of neural networks in enhancing password strength checkers against sophisticated adversarial attacks. We compare our findings with those of de Castro et al., who performed a similar analysis. This paper encapsulates related work, our methodology, and results, exploring the significance of these technologies and techniques in strengthening password-based security systems.**

Youtube Video link: https://youtu.be/yoGmi1pZUHs

I. Introduction

Password security is a critical concern in our digital landscape. Numerous online platforms rely on passwords as their primary authentication mechanism. Despite advancements in authentication protocols, passwords persist as a ubiquitous method due to their simplicity and user familiarity. However, this reliance on passwords poses significant security challenges. Passwords are inherently vulnerable to many attacks, notably brute-force and sophisticated guessing attacks, which have been utilized in several high-profile data breaches (for example, the 2014 iCloud breach that exposed private pictures belonging to celebrities). Incidents like these illustrate the urgency for more effective methods to assess and enhance password strength.

Traditional password strength meters rely on heuristic approaches and assess password strength based on factors like length, complexity, and character variety. Although these heuristics are intuitive and easy to implement, they fall short in realistically predicting a password's resilience against an informed adversary because such meters overlook the probabilistic nature of guessing attacks, where attackers prioritize guessing more common passwords. Consequently, there is a need for strength assessment tools that account for the likelihood of a password being guessed, rather than just its superficial complexity.

Recent advancements in machine learning, particularly neural networks, have opened new avenues in password security. Neural networks, with their ability to learn and predict complex patterns, are well-suited for modeling the probability distribution of passwords. This approach is more aligned with the strategies employed by attackers, who often use probability-based methods for guessing. By simulating an adversarial model, neural networks offer a more nuanced and accurate assessment of password strength, a significant step forward from traditional heuristic-based methods.

Our research is motivated by the study ”Modeling Password Guessing with Neural Networks” by Leo de Castro and colleagues, which demonstrated the use of Long Short-Term Memory (LSTM) neural networks for password strength assessment. Our objective is to replicate and validate their findings. By reimplementing their methodology and conducting a parallel analysis, our study aims to corroborate the effectiveness of LSTM-based models in predicting password strength against guessing attacks. This replication is crucial for affirming the reliability and applicability of neural network-based password strength meters in real-world scenarios.

We evaluate our model by comparing its strength assessments with the assessments outputted by zxcvbn, an open-source password strength estimator designed by Dropbox. It uses pattern matching and conservative estimation techniques to gauge the strength of passwords, taking into account various factors like common passwords, phrases, and behaviors. Unlike traditional strength estimators, zxcvbn provides more realistic results by recognizing and penalizing weak patterns commonly found in user passwords, such as strings of words concatenated together.

This paper provides a comprehensive account of our replication process, methodology, and results. We detail our implementation of the LSTM neural network model and the Monte Carlo technique for guess number estimation. Our analysis not only compares our results with those of the original study but also discusses the implications and potential improvements of neural network-based password strength checkers. Through this replication study, we contribute to the body of knowledge in cybersecurity by offering insights and validation for the use of advanced machine learning techniques in enhancing password security.

II. Related Work

In 2016 the USENIX Security Symposium published a paper called Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks. In this paper, five Carnegie Mellon professors demonstrated how using neural networks for password strength checking is a much more suitable method than current client-side password checkers, which "are often very computationally intensive, requiring hundreds of megabytes to gigabytes of disk space, and taking days to execute"[1]. In contrast, neural networks "guess passwords more successfully than other password-guessing methods in general"[1], returns results in fractions of a second, and can be compressed "into hundreds of kilobytes, which is small enough to be included in an app for mobile devices, bundled with encryption software, or used in a web page password meter"[1]. Like the research paper we are replicating, they make use of neural networks and Monte Carlo simulations, but then go beyond to actually create a "password-guessing model sufficiently compressible and efficient for client-side proactive password checking"[1] with Javascript.

III. Methodology

In this section, we describe the components of our neural network strength checker - the neural network that outputs probabilities for passwords and the Monte Carlo simulation technique used to turn these probabilities into guess numbers and strength scores. The high-level flow that a password follows when being assigned a strength can be summarized as the password is inputted to the neural network, the neural network outputs a probability of that input password occurring according to its probability distribution, the Monte Carlo component uses this probability to determine a guess number by comparing the password's probability to the sampling distribution probabilities, and the guess number is compared to a threshold to assign a strength score to the password.

A. Neural Network

The first component of our neural network strength checker is our neural network, which we use to define a probabilistic model, or probability distribution, for passwords.

Neural networks are computational models that imitate neurons like those in an animal's brain. Neural networks are adept at predicting patterns and generating new patterns when given a large enough sample of training data.

In our research, we used a Long Short Term Memory (LSTM) neural network, which uses a recurrent architecture to "remember" values over some period of "time". Recurrent neural networks are capable of generating sequences that did not appear in its training data, so our neural network is capable of assigning non-zero probabilities to passwords that did not appear in the training data. The input to our model is a string of characters, and the output is a probability distribution of what the next character is predicted to be. We utilized the Tensorflow library created by Google to architect and utilize our neural network, and we performed our entire research project using Google Colab, a browser-based Jupyter-Notebook-style IDE from Google where users can write, execute, and collaborate on Python code.

Our model works at the "character level", using the contextual characters that have already appeared to output a probability distribution of which characters are most likely to appear next. The neural network is able to generate this probability distribution because it has determined the most common patterns based on its training data. For example, if the passwords 'cat', 'cate', and 'cap' appeared in the training dataset, the input 'ca' would output a probability distribution across all possible characters, where the probability of 't' would be larger than the probability of 'p'. In a hypothetical scenario where the only possible characters are 'c', 'a', 't', 'e', and 'p', the outputted probability distribution would show the highest probability for 't', the second highest probability for 'p', and much smaller probabilities for 'c', 'a', and 'e'.

Additionally, we can use the neural network to capture the probability of a particular password occurring by tracking conditional probabilities and multiplying them together. In other words, we can provide a full string (a password) as input and determine the probability that the input string is a password by applying the chain rule for conditional probability. Determining this probability is achieved by iterating through each character in the input password, updating the context, multiplying a running probability (initialized at 1) by the conditional probability of the next character given the context, and continuing until the entire password has been processed, returning the overall probability of that password. For example, if we want to calculate the probability of the string 'ca' being a password, we initialize our running probability to 1. We use the model to determine the conditional probability that the first letter is 'c', given that there are no letters before it, and multiply our running probability by this conditional probability. We then use the model again to determine the conditional probability that the second letter is 'a', given that the first letter is 'c', and multiply our running probability by this conditional probability. Finally, we use the model to determine the conditional probability that the final letter is some end-of-string character, given that the first two letters are 'ca', and multiply our running probability by that conditional probability before returning the final running probability that signifies the probability of 'ca' being a password.

B. Monte Carlo Guess Numbers

Since we want our Monte Carlo simulation to estimate a number of guesses needed to guess a password, we need to be able to sample from the probability distribution that the neural network learned over the entire universe of passwords. We consider a current password as ending when the model predicts the next character to have index 0 in the tokenizer's word_index, and we have the neural network reset the context so that the next character generated is at the beginning of a new password. When generating this sample, it is not required to generate an enormous sample of passwords - generating a small sample of new passwords for the Monte Carlo simulation works just fine.

To generate passwords, we start with an empty context and calculate conditional probabilities for the next character based on the current context. We then sample the next character from the probability distribution and update the context, continuing until we encounter the designated end character that signifies a completed password.

After generating our passwords, we record the probability of each password as a whole (based on the distribution learned by the neural network) and store these password-probabilities key-value pairs in a dictionary. Consequently, when we need to use the Monte Carlo method to estimate a guess number for a target password, we are able to compute the probability of that password, as well as take a sample from the distribution of passwords that the neural network learned.

C. Data

In an effort to reflect the results of our inspiring research paper, we also made use of the rockyou.txt dataset of leaked passwords to train and test our neural network. Fig. 1 shows how accurately the model performed during training and validation, and Fig. 2 shows the model loss during training and validation.

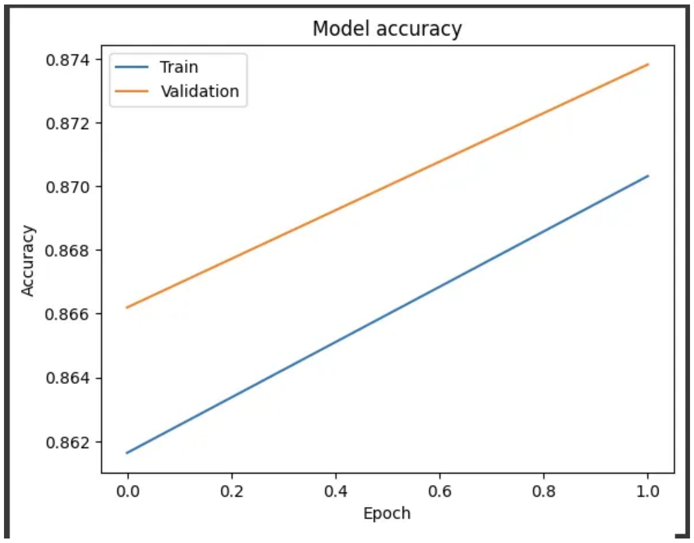


Fig. 1. Model Accuracy

We did have to slightly diverge in the sense of number of passwords used from the dataset due to time constraints and personal device memory limitations, but were still able to train the model with 30,000 passwords, create a password probability distribution with 5,000 passwords, and compare our neural network with the zxcybn method with 1,000 passwords. For the password probability distribution we used 4,000 passwords from rockyou.txt and 1,000 passwords generated by our neural network. For the password strength checker comparison we used 500 passwords from each of the leaked password datasets phpbb.txt and faithwriters.txt.

IV. Results

We evaluated our neural network based strength checker by comparing it quantitatively and qualitatively to the performance of the popular heuristic, zxcvbn. The zxcvbn system can assign a password a strength score ranging from 0 to 4,

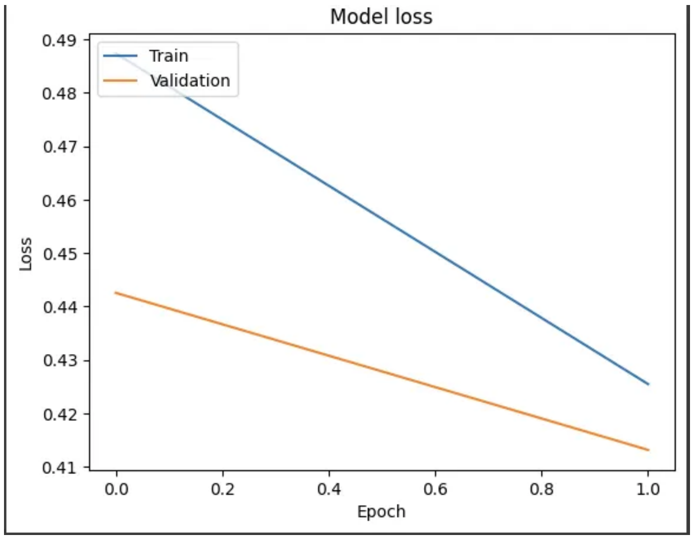


Fig. 2. Model loss

where 0 represents a very weak password and 4 represents a very strong password.

In order to compare our strength checker to zxcvbn, we also assigned passwords a strength score based on thresholds for the estimated guess numbers. To set these cutoffs, we originally tried a cumulative distribution of passwords over guess numbers and determined the cutoff points w, x, y, and z where 25% of passwords had a guess number estimate less than w, 50% of passwords had a guess number estimate less than x, 75 of passwords had a guess number estimate less than y, and 99% of passwords had a guess number estimate less than z, mirroring the approach used by Castro et. al. After analyzing the results, we discovered that our strength checker performed more similarly to zxcvbn when we set these cutoffs by determining the minimum and maximum guess number estimates and splitting our range into 4 equally sized buckets. Regardless of the approach, in the end, we had 5 bins (strength scores) that a guess number estimate could map to, just like zxcvbn.

In order to ensure that our model was generalizing well, we used passwords from the phpBB and FaithWriters password leaks to test our model. We randomly sampled 500 passwords from each of those datasets for a total of 1,000 passwords to use for testing. We assessed the strengths of these 1,000 passwords using both our neural network based model and the zxcvbn model and compared the results to see how similar the strength assessments were. The results of our quantitative analysis are illustrated in Fig. 3 and Fig. 4.

Our model calculated the probability that a given string was a password by combining successive probabilities of characters contextualized by their previous characters, a strategy that hinges on the neural network learning password patterns from the training dataset to a degree where it can predict common patterns. These patterns become more difficult for a network to predict as the strings grow longer, decreasing conditional probabilities and increasing guess numbers. Consequently,

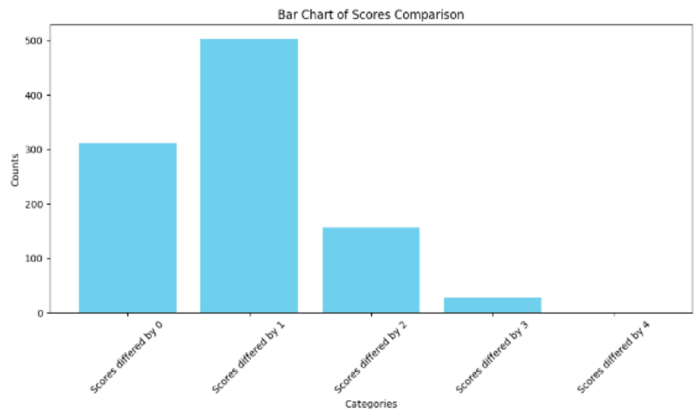


Fig. 3. A graphical representation of the absolute difference in strength scores of the two strength check methods for every password.

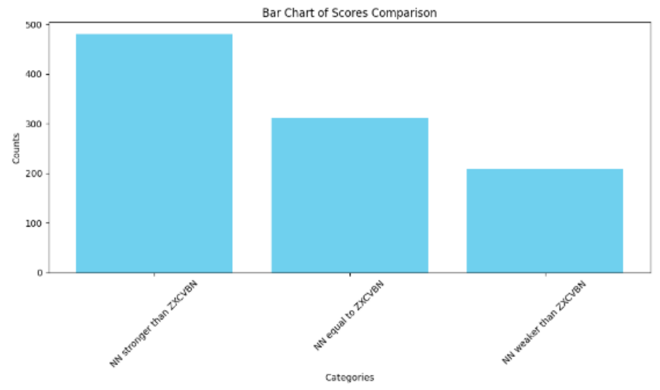
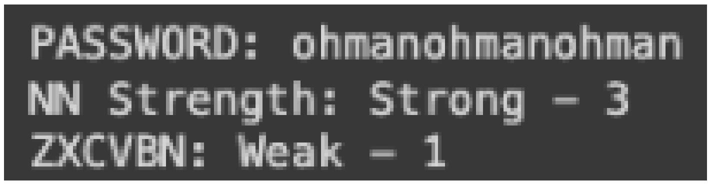


Fig. 4. A graphical representation of how many passwords our neural network scores as stronger than, equal to, or less than zxcvbn.

similar to the work of de Castro et. al, we found that our neural network based strength checker tended to overestimate the strength of long, patterned passwords built from strung-together English words while zxcvbn detected the words and therefore assigned a relatively weaker strength score. Fig. 5 gives an example.



```
PASSWORD: ohmanohmanohman
NN Strength: Strong - 3
ZXCVBN: Weak - 1
```

Fig. 5. Example of a difference between our neural network and zxcybn in relation to password style.

In the zxcvbn model, passwords are evaluated based on how they conform to recognizable structures like words, character-replaced words, and common names, and these types of passwords are perceived as less secure. On the other hand, the neural network model learns from the training data and assigns low probabilities to patterns that did not appear in the training data. This approach is challenged by long passwords, which are less common in training sets and lead to difficulties in predicting their strength.

One way to improve the neural network's performance might be to increase the diversity of its training set. However, this method is not a guaranteed solution because as password standard evolve and longer passwords become the norm, the use of natural-language phrases or sentences will likely increase due to their memorability. Tools like zxcvbn can evolve with this trend by identifying structured passwords of varied lengths, indicating that proxy-based methods like zxcvbn might scale more effectively for longer passwords than current neural network based approaches. For neural networks to accurately predict the complex structure of these longer passwords, a more advanced approach than character analysis is needed, which may involve techniques more sophisticated than the current scope of neural network capabilities.

Conclusion

In conclusion, our research aimed to replicate and validate the findings of the study "Modeling Password Guessing with Neural Networks" by Leo de Castro et al. We successfully implemented a Long Short-Term Memory (LSTM) neural network and a Monte Carlo simulation technique to assess password strength by modeling the probability distribution of passwords.

A. *Limitations of Our Study*

The most significant constraint on our research was the hardware we had access to. Since we were using Google Colab, we were particularly restricted in the contexts of memory and runtime duration, which impacted the scope and depth of our study.

Firstly, the memory allocation on Google Colab is capped, which posed a challenge in handling the large datasets used in our research, in particular the rockyou dataset and our stored probability distribution across the 'universe' of passwords. The memory limits forced us to downscale our data or employ data sampling methods, particularly when training our neural network and when creating our passwords-to-probabilities sampling distribution. While we were still able to proceed with our analysis, these constraints introduced limitations in the robustness of our results.

Finally, another significant limitation was the restricted run-time environment. Sessions in Google Colab are time-bound, and longer sessions, especially those exceeding 12 hours, are automatically terminated. This was a considerable hurdle especially when training our neural network and creating our sampling distribution of passwords-to-probabilities, which affected the continuity and efficiency of our research process.

B. *Future Work*

While Google Colab was a viable platform for our research given the cost and accessibility advantages, the hardware limitations undoubtedly influenced the scope, methodology, and potentially the outcomes of our study. Future research might overcome these challenges by utilizing more robust computational resources, allowing for more extensive data processing and more complex model training, thus potentially leading to more comprehensive insights and findings.

The exploration of hybrid models that combine neural networks with other machine learning techniques could further improve the accuracy of password strength prediction. Integrating behavioral analysis and context-aware features into the model may better simulate real-world password usage scenarios. Continuous monitoring and adaptation of the model to emerging password patterns would also be essential to stay ahead of evolving security threats.

As seen in [1], a neural network password strength checker has the ability to be developed into a fast and effective usable password checker on any kind of device, and that is the next big step for our project.

Future enhancements could also include the integration of natural language dictionaries, like common names and English words, which have improved performance in Probabilistic Context-Free Grammars (PCFGs). Another potential improvement would be training the model with mangled strings, resembling human password creation habits. Our current model only assessed password strength, but did not provide any guidance on creating a stronger password or insight into why the current password was deemed weak.

Our study also did not explore the nuance between various password policies, such as minimum length or character requirements. Tailoring the model to specific password classes could have resulted in more accurate strength estimates. However, gathering data for these policies can be challenging.

Providing instant password strength feedback is powerful, but neural networks still face challenges like memory consumption. Techniques like weight quantization can compress the network, allowing for browser deployment. Melicher et al.'s Javascript implementation exemplifies this approach[1]. It's critical to consider that user adoption hinges on intuitive interfaces, exemplified by the development of user-friendly applications like zxcvbn's interactive tool.

References

[1] W. Melicher et al., "Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks," 2016. Available: https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_melicher.pdf

[2] Dell'amico, Matteo, and Maurizio Filippone. "Monte Carlo Strength Evaluation." Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15 (2015).https://dl.acm.org/doi/10.1145/2810103.2813631

[3] Wheeler, Daniel Lowe. "zxcvbn: Low-Budget Password Strength Estimation." Proceedings of the 25th Usenix Security Symposium (2016). https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_wheeler.pdf