Better passwords through science (and neural networks)*

William Melicher, Blase Ur, Sean M. Segreti,
Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor

We discuss how we use neural networks to accurately measure password strength, and how we use this capability to build effective password meters. First, we show how neural networks can be used to guess passwords and how we leveraged this method to build a password guesser to better model guessing attacks. We report our measurements of the effectiveness of neural networks at guessing passwords, demonstrating that they outperform other popular methods of modeling adversarial password guessing. We then show how we developed a password guesser that can be compressed so that it is practical for client-side use inside a web page [1]. Finally, we describe how we designed and built a password meter, based on neural networks, that gives more accurate and helpful guidance to users for creating passwords that are resistant to guessing attacks [2].

Passwords are the most common authentication mechanism in use today. We all use passwords every day and will likely continue to do so for the foreseeable future. Unfortunately, human-chosen passwords often follow predictable patterns. For example: exclamation points are at the end; capital letters are at the beginning of passwords; dictionary words, well-known phrases, keyboard patterns, and names of people and places are all common. Such predictable patterns allow attackers to break into accounts by guessing passwords.

Guessing attacks can take the form of online attacks in which attackers make guesses while trying to log in to a live system. Online attacks are sometimes defended against by limiting the rate at which attackers may make guesses against the system. In contrast, in offline guessing attacks, attackers can make large numbers of guesses without limits. This commonly happens when a database of hashed passwords is stolen, an event that occurs with disappointing regularity. Attackers guess candidate passwords and compare them against hashed passwords in the database, limited only by the amount of computer resources they have. The widespread incidence of password reuse makes such attacks more dangerous because attackers who crack a user's password that was leaked from a stolen database may use that cracked password—or common variations of the password—to guess the credentials for that user's other accounts. A common and effective defense against both online and offline guessing attacks is to urge users to create less predictable passwords that are more resistant to guessing.

To understand how to guide users to make less guessable passwords, our research group has studied methods for modeling how attackers guess passwords. Previous approaches for modeling password-guessing attacks include statistical approaches, and tools used in adversarial password cracking. Statistical methods, such as Markov models and probabilistic context-free grammars, work by deriving statistical properties from lists of training passwords. Adversarial password cracking tools, such as John the Ripper and Hashcat, are typically used in practice for their ability

*This is the authors' version of this article; the official version appears in USENIX ;login: Winter 2017, Vol. 42, No. 4 (https://www.usenix.org/publications/login/winter2017/melicher).

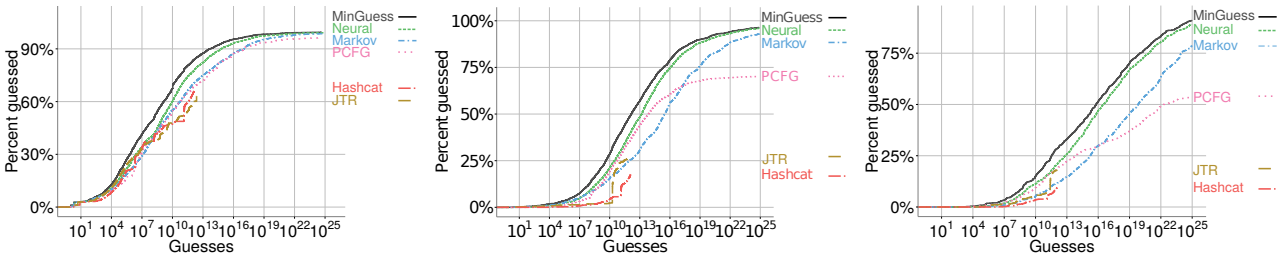
to crack hashed passwords quickly; often they are configured by experts to craft special password cracking rules for specific password sets. Prior work from our group has studied these approaches, and shown how the combination of multiple automated approaches approximates the ability of professional human experts to guess passwords [3]. However, modeling a guessing attack in which attackers can make large numbers of guesses often requires servers with tens of CPU cores and with gigabytes of disk space for storing models of password guessing. Such models are not practical for giving real-time feedback to users during password creation; users can't download gigabytes of data or wait days or weeks to get feedback for creating a password.

Due to the challenges of accurately modeling password attacks, most password meters are unable to provide data-driven, principled feedback to users during password creation. Meters will typically calculate some combination of a variety of heuristics—such as the number of special characters used or the length of the password—which often has little correlation to the resistance of passwords to guessing attacks [4]. When faced with such meters, users often make predictable modifications in order to satisfy the meter's strength estimate, such as adding an exclamation point to the end of their password. However, because attackers are also aware of the predictable patterns in password construction, such modifications do little to improve the password's resistance to guessing. In addition, meters are often incapable of providing positive advice or giving users suggestions about how to make passwords better, instead rating a password as simply "weak" or "fair."

Design of a neural network guesser

Neural networks are a machine-learning technique that is particularly adept at fuzzy classification problems and problems dealing with computer processing of natural language. The intuition for our approach was that, because the task of guessing passwords in an adversarial attack is conceptually related to generating natural language, neural networks would be well suited to our goal of modeling guessing attacks. Recently, the machine-learning community has showed how to use neural networks to generate text, which our approach leverages [5]. Generating a password with a neural network involves repeatedly predicting the next character of a password to build up the password one character at a time. This process can be extended to generate large numbers of probable passwords. During training, the neural network is taught to predict the next character when given a real password fragment. The neural network can then learn to recognize high-level patterns that often arise in password construction, such as keyboard patterns or exclamation points at the end of a password.

We tried many different variations and tunings for training our neural network guesser. When training neural networks, there is a large design space of different parameters and design decisions to explore for better performance. We experimented with a wide range of different parameters including: the number of parameters in the model; the method of representing password characters; different recurrent neural-network architectures; using different types of training data; and using a technique called transference learning, which specializes neural network predictions for different situations. At the end of these experiments, we had a neural-network training methodology that we found was most accurate for our application of guessing passwords. Additionally, we used a technique of modeling password guessing to arbitrarily high numbers of guesses by employing Monte Carlo methods [6], allowing us to accurately model password guessability against nation-states or other extremely powerful adversaries who have huge resources for cracking passwords.



(a) Guessing passwords that must be more than eight characters.

(b) Guessing passwords that are required to be more than eight characters long and have a mix of character classes.

(c) Guessing passwords that are required to be more than 12 characters long and have a mix of character classes.

Figure 1: Comparison of the ability of different password methods to guess passwords. The x-axis of each graph shows the number of guesses made in log scale. The y-axis shows the percent of passwords guessed. Higher lines on the graph represent more accurate guessing. "Neural" shows the performance of our neural-network approach; "Markov" the Markov model approach; "PCFG" probabilistic context free grammars; "JTR" John the Ripper; "Hashcat" shows the performance of Hashcat; and "MinGuess" shows a combination of all approaches, where a password receives the minimum guess number from all approaches. Each graph shows passwords created under a different policy—requiring a different minimum length and different mix of character classes (uppercase and lowercase characters, digits, and symbols).

When designing our neural-network guessing method, we tested it against the best tunings of other methods for guessing passwords. In addition, during development of our neural-network guesser, we comprehensively tested various different versions of the neural-network guesser against each other to find the best method. We measured the performance of our guessing approaches both on real passwords collected in recent password leaks and on passwords we have collected in our research studies, allowing us to compare the performance of guessing methods in a wide variety of password policies and situations. To train our guessing methods in our experiments we require large numbers of real passwords, which we obtained from leaked password lists. In total, our dataset of passwords contained over 100 million passwords from more than 20 password leaks. This huge amount of data on real-world passwords allows machine-learning techniques to infer deep insights into password construction and to have the predictive power to model common password patterns.

We found that the neural networks guessed passwords more accurately than any other individual method for guessing passwords. However, while our best performing neural networks often performed close to an optimal guessing strategy, the combination of all methods including neural networks (MinGuess in Figure 1), performed better than just neural networks alone, showing that a combination of many models is still better than any individual method. Nonetheless, if one is limited to only one method for estimating password strength, neural networks are the most accurate. Figure 1 shows a selection of some of our results on guessing accuracy for different conditions; the neural network approach guesses a larger proportion of passwords over the same number of guesses as other methods. This finding holds to various degrees across all of our test sets; although, we find that neural networks are particularly accurate when guessing passwords made under the more exotic, stronger password policies, which are becoming increasingly common as password guessing abilities increase.

Design of a client-side password strength estimator

Besides increasing the accuracy of existing password strength models, we also strove to develop more practical models. Previous methods for modeling adversarial password cracking require large amounts of disk space or bandwidth—hundreds of megabytes or gigabytes—and take hours or days to calculate measures of password strength. In contrast, to give real-time feedback to users during password creation, models must be smaller to download and give quick results. For this application, we wanted a model that was less than one megabyte to download, which is roughly half the size of an average web page. Additionally, in the context of real-time feedback, a model must calculate a measurement of password strength within a fraction of a second—ideally below the threshold of human recognition, which is roughly 100ms. In addition to these properties, the measurement should be accurate, and the model should run inside of a web browser, which means that JavaScript is the most viable execution platform.

Given the challenges of implementing accurate password-strength measurement on resource-constrained clients, it might be tempting to use a system architecture where the password model is stored on a server and only measurement results are communicated to the client. However, in many situations the user's password should never be sent to the server for security reasons, for example, in the case of device encryption software, keys that protect cryptographic credentials, or the master password for a password manager. Even in cases where the user's password is eventually sent to an external server, using a remote password-strength measurement mechanism may allow powerful side channels based on keyboard timing, message size, and caching [7]. For these reasons, we preferred architectures where password modeling and strength estimation are done entirely on the client side. This design decision has the added benefit of being easier for web administrators to deploy.

To summarize our technical approach to meeting these goals: We started by training a neural network with fewer parameters—the features of the model that define how to predict the next character. Using this less complex model made the network smaller, but did not sacrifice much accuracy compared to our best-performing network. Then, we reduced the precision of the already shrunken neural network's parameters, again trading off space for some accuracy. Finally, we used standard lossless compression methods to further shrink the size of the model, eventually reaching a model size of 850KB. To make our network produce low-latency results, we pre-computed an approximate mapping for estimating the strength of the password, which is sent to the client along with the network. In addition, we cached specific intermediate computations, so that the common case, in which a character is added to the end of the password, is quicker because the strength estimator only needs to update its previous computation. We were able to get the average response time to be 17ms for this common case. Some of our optimizations sacrificed accuracy for the sake of quicker results or a smaller model; we empirically measured the impact that such optimizations introduced and found the error rate to be small enough to be acceptable for our purposes. In addition, we tuned the network so that it was much more likely that we would make safe errors—underestimating a password's strength—than unsafe errors.

We compared the accuracy of our client-side strength estimation based on neural networks to existing password meters: zxcvbn and Yahoo's password meter. zxcvbn, in particular, measures password strength using a number of highly tuned heuristics for password strength. We found that our method of measuring password strength to be more accurate—correlating more highly with password strength measured by simulating a guessing attack—than either meter, having between 39% and 83% fewer unsafe errors, depending on the meter and the password policy.

At the same time, our strength measurement also had fewer safe errors. In addition, our more principled method of simulating adversarial guessing entirely on the client-side has the benefit that it can be easily reconfigured—by re-training the neural network—for new password policies or new situations. We know that certain password sets often have special patterns that are unique to that set, for example, passwords for a sports website may contain more sports terminology than other password sets. Our method would be able to be easily retargeted to learn such patterns.

Design of a password meter

While the development of an accurate client-side strength-estimation tool is necessary for a password meter, it is not sufficient. There is a gap between a practical measurement of strength and providing effective real-time feedback about how to make a better password. We wanted to bridge this gap. Our main goal was to give human-understandable feedback about password creation; our neural-network strength estimation by itself can tell the user that a password is weak or strong, but it cannot say how to improve the password to be more resistant to guessing. To accomplish this, we aimed to give two types of suggestions: First, we wanted to be able to provide concrete suggestions for specific passwords that are stronger. Second, we wanted to provide high-level guidance to users that is specific to their exact situation, for example, notifying users that using capital letters at the beginning of the password is a common pattern, and does not meaningfully improve the strength of their password.

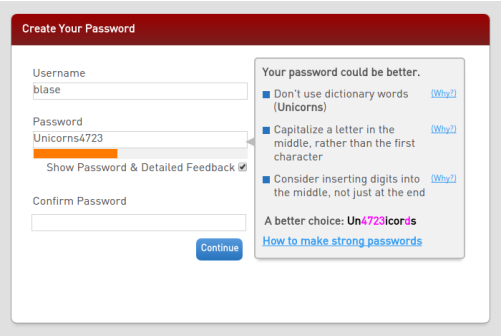


Figure 2: Screenshot of our password meter's interface. The bar shows the strength estimate of the user's password. The popup dialog shows specific password feedback based on the user's password.

We developed a password meter that achieves these goals. Our meter combines the accuracy of our neural-network strength measurement with a series of data-driven heuristics that provide human-understandable feedback about the user's password. Figure 2 shows an example of our meter in use. Our meter uses the neural network to control the bar that shows how strong the user's password is, while using the data-driven heuristics additionally give the user specific feedback about how to improve their password. The meter can also provide a concrete suggestion for how to change the password so that it will be stronger. It does so by creating several candidate suggestions that are similar to the user's chosen password and then using the neural network to gauge their strength. Only those candidate passwords that are judged stronger by the network are shown to the user.

We tested whether the meter helps users to create stronger passwords. We recruited participants to create a password for a hypothetical high-value, online account in a variety of different conditions—some participants used our meter during password creation, some used modified versions of our meter, and some did not have the benefit of any meter. Similar methodology has been used in prior work by our group for measuring the impact of a variety of different conditions on the security and usability of human-chosen passwords [8, 9].

We found that participants who used the meter created passwords that were 44% more resistant to

guessing attacks than those who did not. Interestingly, we also found that participants who saw the human-readable suggestions produced even stronger passwords than those who only saw the measurement of strength. This implies that not only does providing real-time strength estimates help users make stronger passwords, but also that providing actionable suggestions about what users should do provides additional benefit.

Conclusion

We showed how neural networks can be used to guess passwords, and that they can do so more accurately than other methods for adversarial password guessing. We also showed how leveraging neural networks can lead to more practical estimations of password strength on resource-constrained client machines in real time. Finally, we built and tested a password meter, based on neural networks, that gives human-understandable feedback and guides users to make better passwords. We have released our meter as open source software (at `https://github.com/cupslab/neural_network_cracking` and `https://github.com/cupslab/password_meter`), and invite people to use it.

Acknowledgments

We would like to thank Mahmood Sharif for participating in discussions about neural networks and Dan Wheeler for his feedback. This work was supported in part by gifts from the PNC Center for Financial Services Innovation, Microsoft Research, John & Claire Bertucci, and a gift from NATO through Carnegie Mellon Cylab.

References

[1] William Melicher, Blase Ur, Sean M. Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. "Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks." In *Proceedings of 25th USENIX Security Symposium*. 2016.

[2] Blase Ur, Felicia Alfieri, Maung Aung, Lujo Bauer, Nicolas Christin, Jessica Colnago, Lorrie Faith Cranor, Harold Dixon, Pardis Emami Naeini, Hana Habib, Noah Johnson, and William Melicher. "Design and evaluation of a data-driven password meter." In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 2017.

[3] Blase Ur, Sean M. Segreti, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Saranga Komanduri, Darya Kurilova, Michelle L. Mazurek, William Melicher, and Richard Shay. "Measuring Real-World Accuracies and Biases in Modeling Password Guessability." In *Proceedings of the 24th USENIX Security Symposium*. 2015.

[4] Xavier de Carné de Carnavalet and Mohammad Mannan. "From Very Weak to Very Strong: Analyzing Password-Strength Meters." In *Proceedings of the 18th Network and Distributed System Security Symposium.* 2014.

[5] Ilya Sutskever, James Martens, and Geoffrey E. Hinton. "Generating text with recurrent neural networks." In *Proceedings of the 28th International Conference on Machine Learning (ICML-11).* 2011.

[6] Matteo Dell'Amico and Maurizio Filippone. "Monte Carlo strength evaluation: Fast and reliable password checking." In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security.* ACM, 2015.

[7] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. "Timing Analysis of Keystrokes and Timing Attacks on SSH." In *Proceedings of the 10th USENIX Security Symposium.* 2001.

[8] Saranga Komanduri, Richard Shay, Patrick Gage Kelley, Michelle L. Mazurek, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Serge Egelman. "Of passwords and people: measuring the effect of password-composition policies." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* ACM, 2011.

[9] Richard Shay, Saranga Komanduri, Adam L. Durity, Philip (Seyoung) Huh, Michelle L. Mazurek, Sean M. Segreti, Blase Ur, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. "Can long passwords be secure and usable?" In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems.* ACM, 2014.