Different Approaches to Memorable Password Generation

Tatsuya Aoyama | Ryan A. Mannion | Nitin Venkateswaran | Zihao Ye

Abstract

This project aims to generate strong, memorable passwords using techniques from natural language processing, inspired by recent changes in Georgetown University's password policies. Three main approaches are taken: (1) a rule-based system with user input, (3) language model based approaches, and (4) an autoencoder system.[1] The strength of each system is evaluated quantitatively against the Georgetown baseline, and the memorability is qualitatively discussed. No single system turns out to be the best system in all aspects, but a recommendation is made based on practical considerations, and implications for future work are discussed.

Introduction

User-generated passwords are often more memorable, but their security level is sometimes overlooked or compromised, as the use of birthdays, nicknames, or other predictable pieces of information to generate passwords can lead to serious security risks. However, automatically generated passwords are often random sequences of characters, numbers, or unrelated words, which adversely affect memorability. This project aims to tackle this memorability-security tradeoff problem.

Related Works

User-generated passwords are rarely exceedingly secure. Users without password managers must memorize passwords for many accounts, which often leads to the use and reuse of easy-to-remember passwords. Bonneau and Shutova (2012) show that humans are not unpredictable in their password generation methods, leading to security flaws. Sites which generate passwords for their users might use a template system, which generates random strings to meet a set of guidelines (e.g. one capital letter, one number, a symbol, etc.). These passwords, while possibly more secure than a user-generated one, are often hard to remember especially if satisfying a minimum-length constraint. (Komanduri et al. 2011). Other studies show the efficacy of mnemonic password generation techniques, whereby users think of a sentence and use substrings from each word to form a more random-looking yet memorable password. (Yan et al. 2000, Yan et al. 2004, Yang et al. 2016)

[1] Code for all experiments is available at github.com/ryanamannion/gtown-passwords

There seems to be some evidence that even short (i.e. two-word) passphrases could improve security over weak passwords: Bonneau and Shutova (2012) estimate an increase from 10 bits of entropy to 20 bits with just a two-word passphrase. That number increases to 30 bits for a four-word phrase. Providing users with a passphrase thus seems like a sensible option to increase low-end password security without the inconvenience of hard to remember passwords.

Experiments

Baseline

The goal of the baseline is to mimic Georgetown's new password generation system as closely as possible. To that end, we consulted Marty Johnson, Chief Engineer at Georgetown University Information Services (UIS) and one of the people responsible for implementing the new password policies. UIS's primary goal for the new system was to increase password strength without making them impractical for users to remember and type, presumably since many users will type the passwords on mobile keyboards, where switching between letters and symbols can be frustrating.

The basic structure of Georgetown's new passphrases is as follows:

- 15 < length < 32
- Five words
- Separating symbol (can be ., -, @, !, or None)
- First letter of each word capitalized (optional)

The words are selected from a DiceWords list: a wordlist where words are paired with a number representing a roll of n dice read from left to right. While the exact wordlist UIS uses was not disclosed, we were told that UIS uses a subset of a freely-available wordlist that was constructed by removing homophones, words which are commonly misspelled or rare, and vulgarity. The resulting passphrases are reported to have increased the average entropy from 45 bits to 64 bits.[2]

Our system is implemented in Python 3.8[3] and makes use of the Electronic Frontier Foundation's (EFF) long word list, a freely available wordlist with problematic words involving homophones, rare words, and vulgarity removed.[4] The script works by reading a configuration file with information such as the lower and upper bounds of password length, how many words to generate (w), the separating symbol, and capitalization. Once the user preferences are loaded, five dice are rolled with `random.randint(1, 6)` and the word associated with that number is added to the stack. Once w words are chosen, the total length including any separators is checked against the user-specified lower and upper bounds. The function begins again and will

[2] Marty Johnson, email to authors, April 20, 2021.
[3] Python Software Foundation. Python Language Reference, version 3.9. Available at http://www.python.org
[4] For more information and a link to the wordlist, see https://www.eff.org/nl/deeplinks/2016/07/new-wordlists-random-passphrases.

run until a satisfactory password of right length is generated, or a maximum recursion depth is achieved. The latter should not occur with proper parameters.

Other lists can be configured, to support system agnosticity, as long as the list is of the same format. The resulting passwords resemble those generated by Georgetown's in-house system, and the evaluation shows they perform similarly with respect to entropy. An example of a password generated by this system is *wronged.monsoon.relearn.important.counting*.[5]

Incorporate Semantic Information from User Input

A user-defined keyword is used to enhance the memorability of the auto-generated password. A simple implementation of this idea involves the user providing a preferred keyword and a generator to find candidate words based on their semantic relevance to the keyword, which are then used in the password generation process. For example, if the user-defined input is "baseball," relevant candidate words, such as "base," "homerun," or "pitcher" are found. We experimented with two approaches: word embeddings and WordNet (Princeton University, 2010).

GLoVe (Pennington et. al, 2014) 300-dimensional pre-trained word embeddings are used. Semantic relevance is operationalized as cosine similarity. Although the initial plan was to randomly extract, based on cosine similarity, *k* semantically related words from the top *n* words with highest relevance to the keyword, it was considered computationally too expensive across all 400,000 words. A workaround was to randomly select a smaller random sample of 1,000, and output *k* similar candidates from that subset. Although this approach made the computation much more feasible, the randomness affected the similarity with the user-defined input keyword, compromising memorability. As another alternative, we ran K-means clustering to generate 200 – 1,000 clusters, only considering the words in the same cluster as the keyword. The algorithm did not converge after 24 hours with the limited computational resources available, and a computationally more efficient approach became necessary.

WordNet is a collection of semantic concepts (i.e., synsets) associated with corresponding lemmas. These concepts are hierarchically related to each other in a tree-like structure (e.g., as synonym, hypernym, and hyponym). A random walk search of the tree across concepts present at the same level or at lower levels ,i.e. all hyponyms, is used to search for related words. This is based on the intuition that words that are more specific are easily associable with the original word than words that are more abstract. For example, given the user-defined input *baseball*, *homerun* (hyponym) is considered intuitively more memorable than *sports* (hypernym). Once all relevant semantic concepts are identified based on the above inclusion criteria, all lemmas associated with each of the concepts are added to the list of candidate words. From this list, *k* random words are selected as final candidates to be included in the password.

To further strengthen the password while balancing memorability, a second-language feature is added. Users also indicate their preferred second language, and the algorithm described above is

[5] More examples can be seen at https://github.com/ryanamannion/gtown-passwords/blob/main/baseline/examples

repeated but with the lemmas in the selected second language. All such words are Romanized (e.g., 野球 -> yakyuu) and might not directly result in an improved entropy value; however, it was expected to add further security especially against dictionary attacks.

Once the semantically relevant words are selected based on the WordNet-based algorithm, random replacements with numbers and special characters are applied. Following Glory et al. (2019), to retain the memorability of the password, only particular alphabets that resemble numbers (e.g., E -> 3; o -> 0) or special characters (e.g., a -> @; i -> !) are replaced. Although this is purely rule-based, the type of replacement is stochastically determined. For example, with the input word "baseball", second language "spanish," and k = 2, the system returns "beisbol" (baseball in Spanish) and "daisycutter," which were combined as "bE!sb0|-daisycutT3&."

Language Models

The aim with the Language Modeling approach is to stochastically generate strings with some relationship to enhance memorability.

Three alternatives are built: a uniform probabilistic model, a probabilistic model based on the semantic distance between bigram pairs, and an LSTM (Hochreiter & Schmidhuber, 1997) language model. The uniform and the semantic-distance model populate three bigram models, each: one for adverb-adjective pairs, one for adjective-noun pairs, and one for noun-noun pairs. Probabilities are initialized uniformly in the uniform model, and are based on the semantic distance between bigram-pair words in the semantic-distance model, biasing the model towards unlikely yet related bigrams. The cosine distance is calculated using gensim (Rehurek & Sojka, 2011), based on GLoVE (Pennington et.al, 2014) word embeddings. The LSTM model is a single-layer uni-directional LSTM model with dropout regularization, trained using the PyTorch framework (Paszke et. al , 2019). Word embeddings are initialized and trained from scratch.

All alternatives generate passphrases using four templates: adverb-adjective, adverb-adjective-noun, adverb-adjective-noun-noun, and adjective-noun. These templates were chosen for memorability reasons as, intuitively, it may not be possible to generate ungrammatical sequences. The sequences are generated using the bigram models described and then concatenated; with the LSTM model, the sequences are extracted post-hoc after generating full sentences. The dataset is a collection of works by Edward Lear and Lewis Carroll (more details in the code repository). The dataset contains unique words invented by the authors, such as 'ombliferous', which could improve memorability.

The three alternatives are evaluated on sets of 1000 passwords, using an independent Kneser-Ney interpolated model from the NLTK package (Bird et. al, 2009), trained on the dataset. The Shannon Entropy (Shannon & Weaver, 1949) in bits is calculated for all alternatives. The LSTM model does the worst, with an average entropy of 9.70 bits as it tries to model actual language patterns; this compromises security as less information is needed to model the password. The semantic-distance alternation with average entropy of 14.39 bits, does not do much better than random bigram generation with average entropy of 14.24 bits.

Autoencoder

The autoencoder model includes an encoder and a decoder, where the encoder captures significant information by mapping original passwords into lower dimension vectors, and the decoder uses the information to recreate the original passwords. The intuition is to use the encoder to learn the patterns (e.g., capital letters, English words, numbers, special characters) in the training data and then use the decoder to generate new passwords with similar patterns, by minimizing the difference between the original passwords and the reconstructed ones.

The dataset used contains 1000 passwords[6]. Each password has 4 parts: words, numbers, special characters and capital letters. These features increase strength and satisfy password constraints in systems., All the passwords are in a specific format, *capital letter* + *English word* + *number* + *special symbol*, making it easy for users to remember and for programmers to augment the data.

Three BiLSTM layers are included in both the encoder and the encoder, connected by a Dense layer of size 6. Passwords are converted to one-hot character encodings. The evaluation focuses on: (1) similarity of original and reconstructed passwords during training, (2) whether the new passwords generated by the model include all the patterns mentioned above. Although the accuracy of the reconstruction task reached 94%, most of the new passwords generated by the decoder did not include all the patterns (e.g., *ccckerm!*, *Ahayk9nnaaa*).

The encoder maps passwords to a smaller latent space, trying to capture features. An issue could be that the simple auto-encoder always tries to minimize the loss while sacrificing good organization of features in the latent space. To solve this problem, we use a variational autoencoder (VAE) whose training is regularised to ensure that the latent space has good features that help the decoder to generate data. The 4 steps of the implementation are: (1) encoding the input as distribution over the latent space, (2) sampling a point in the latent space from that distribution, (3) decoding the sampled point and computing the reconstruction error, (4) backpropagating the reconstruction error through the network. The resulting reconstructed passwords have accuracy of 83%, which is lower than the standard version autoencoder, but includes most of the required patterns.

Although the VAE model does generate good results, the passwords are too simple to be cracked (see Crackability section), and this VAE model can be easily replaced by an unsupervised rule-based system, since all the passwords are in a fixed format. We still tried to improve this model by augmenting the data, and training the model with a different dataset.

To augment the data, we used the same patterns and the format of the passwords in the original dataset, but with more capital letters. The Electronic Frontier Foundation's (EFF) long word list provided an extra augmentation base, increasing the dataset size to 15540 passwords. The training converged after ~8 epochs, with accuracy of 43% with the augmentation ablation. We also trained the VAE with the extra EFF long word list data, with accuracy of 30%. It seems that the VAE model, which gets a 94% accuracy using the original 1000 passwords dataset, is not

[6] For more information about the dataset, see: https://towardsdatascience.com/generating-passwords-with-generative-models-from-probabilistic-to-deep-learning-approaches-54d41d8810e3

suitable for training on these two datasets. Given that it takes a long time to adjust parameters and to train on a large dataset (e.g., the one with 15540 passwords), we will include them as part of our future work.

Results

Two entropy metrics are used: the password entropy, which is the log (base 2) of the number of possible combinations that the system's symbol-set allows, and the binary entropy i.e the approximate entropy of a finite binary string as given in Croll (2013). Static word-lists are used for the baseline and language models, and a static character set for the auto-encoder model. A combination of a static list of 1000 common nouns and characters is used for the user-input model, given its use of random-walk search for semantically related words.

Table 1 in Appendix A details the entropy evaluations of each experiment.

The baseline system shows two different scores: simple and complex. Simple is a collection of passwords with no separators and no capital letters, whereas complex includes all possible combinations of options. The purpose of separating the two is to evaluate the hypothetically "weakest" password that can be generated by the system. These resulting scores were about the same, which is to be expected given that they share the same wordlist. Overall, the baseline matched Marty Johnson's testimony that passwords created with the Georgetown system achieved approximately 64 bits of entropy.

For the User Input model, *Original*, *Simplified*, *5-word simple*, *5-word* complex versions were tested, and they represent the original Glory et al. (2019) implementation (i.e., 2-word with capitalization, number, special character replacements), its simplified version (i.e., 2-word with only capitalization), simplified GU-compatible version (5-word, lowercase and no joining characters), and complex GU-compatible version (5-word, uppercase with joining characters). The best model was the last one (i.e., complex GU-compatible version), which achieved the mean score of 274 and 0.61 for password entropy and bi-entropy, respectively.

The Language Model figures are taken only for the longest generated passwords for the best-performing semantic-distance model. The password entropy is ~41 bits, and the average binary entropy 0.63 for this model.

A standard version and a variational version of the autoencoder are included in the table. The VAE model regularises the latent space to generate better new passwords using the decoder. Both of these two models were trained on the original 1000 passwords dataset. Given the other two versions that trained on the 15000 passwords dataset and GU passwords dataset respectively have not yet been tuned, they were not tested. Although the standard autoencoder's results are slightly better than VAE's, the passwords generated by the standard one neither satisfy the constraints nor capture all the patterns in the training data. Overall, the VAE model performed better, which achieved the mean score of 49.22 and 0.91 for password entropy and bi-entropy respectively.

Discussion

Pros and Cons of Each Model

Crackability

Even if the system outputs have high entropy values, the system is not fully secure if attackers can imitate the system's algorithm. The Autoencoder model might therefore be highly vulnerable, as it essentially learns the pattern present in the training data. Moreover, the sample outputs are highly homogenous, attackers may be able to imitate them if the training data are leaked. For example, once the sample pattern, *capital letter + English word + number + special symbol*, is identified by the attacker, a fewer number of brute-force attacks are needed, which results in significantly compromised security.

The GU Baseline and User Input models may be more robust because of the random selection of words. Even if the attacker obtains the wordlist used in these systems, the number of possible combinations is still extremely large, and capitalization and special character options enhance robustness. As the GU Baseline model, unlike the User Input model, is not constrained to a similar set of words, it may be the most robust.

Language Modeling approaches are not difficult to crack once the dataset, templates, and model types are known.

Ease of implementation

In terms of the computational cost and the dataset required for each system, GU Baseline model is clearly the winner as no training is required. The User Input model also relies only on publicly available datasets and does not involve any training, however the cosine-similarity based search algorithms are computationally expensive. Moreover, if a user selects an infrequent word and a second language that is not strongly supported by WordNet, the system might not find a sufficient number of relevant words. The autoencoder system, on the other hand, requires a unique list of well-defined homogeneous passwords so that the model can learn the password patterns. A well-defined pattern would mean that a similar system could be constructed in a purely rule-based approach. Given the type of the dataset and training required for this model, this simple rule-based alternative might be more practical. Language modeling approaches range in complexity from bigram to neural models, and could be classified either way.

Memorability

Obtaining quantitative measures of memorability was unrealistic given the time constraint, so a qualitative comparison of memorability is given below. Passwords generated by the GU Baseline model might be the least memorable, although some outputs, such as "*correct horse battery staple,"* seem to help us memorize pictorially as a peculiar scene. Notably, the passwords from the GU Baseline model are constrained so that the users do not have to switch the keyboards back and forth on their smartphones. Many of the outputs in the User Input model are more memorable; for example, words in this passphrase generated by the 5-word-variant of User

Input model (see System section), "*Eiki*Akki*Intrepidity*Diimon*Tamashii*," are all related to spirituality, which makes it associatively memorable—even more so given that the theme is user-defined. The original version of the User Input model uses only 2 words and those passwords are arguably even more memorable although they come with the number and special character replacements (e.g., scheme_ryakui -> sCh3me_&yakUi). The autoencoder model also seems to have reasonable memorability thanks to its clearly defined pattern, despite the lower password strength. Language Models are memorable in their capacity to model human language patterns, at the expense of security as reflected in a lower Shannon entropy for the password.

Recommendation

We recommend using the User Input model given the overall balance of its ease of implementation, strength, and memorability. While maintaining some degree of memorability by making the sub-components semantically related to the user input, even the two-word version gives the security level as high as GU Baseline models. One caveat is that the entropy values might have been inflated because of the non-English language words in the passwords; however, this was part of the motivation to include foreign words: to prevent dictionary attacks.

Limitation and Future Work

As mentioned above, memorability is a highly subjective construct, and needs a psychological experiment to obtain a reliable measure, which was beyond the scope of this project. For future work, two possible experiments were considered: (1) ask the human participants to rate the memorability of the password on 7-point Likert scale; and (2) ask the human participants to try memorizing the password, work on distractor tasks, and test if they still remember the password immediately after the tasks (posttest) and after certain amount of time (delayed posttest). Although the latter is more expensive, it should provide a better approximation of the password memorability.

Appendix A - Results Table

Table 1. Descriptive Statistics for Entropy

Model	**Version**	**Password Entropy**				**Bi-Entropy**			
		Mean	**SD**	**Min**	**Max**	**Mean**	**SD**	**Min**	**Max**
GU Baseline	Simple	64.59	1.22	25.84	64.62	0.63	0.34	1.48e-5	0.95
	Complex	64.64	0.41	64.62	77.55	0.64	0.33	5.75e-8	0.95
User Input	Original	94.34	33.32	19.93	231.32	0.63	0.33	1.47e-5	0.95
	Simplified	73.57	35.20	19.93	253.99	0.62	0.34	5.74e-8	0.95
	5-word simple	186.74	40.06	53.96	413.46	0.61	0.33	1.47e-5	0.95
	5-word complex	274.47	74.54	92.66	726.55	0.61	0.34	8.77e-13	0.95
LM	Uniform	41.40	0	41.40	41.40	0.63	0.33	4.6e-5	0.95
	Semantic-Distance	41.40	0	41.40	41.40	0.62	0.34	1.47e-5	0.95
	LSTM	41.40	0	41.40	41.40	0.63	0.34	9.7e-5	0.95
Auto-Encoder	Standard	60.39	14.88	15.63	78.14	0.91	0.34	0.18	0.98
	Variational	49.22	14.64	24.53	73.60	0.91	0.33	0.04	0.97

References

Bird, Steven, Ewan Klein, and Edward Loper. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, Inc., 2009.

Bonneau, Joseph, and Ekaterina Shutova. "Linguistic Properties of Multi-Word Passphrases." In *Financial Cryptography and Data Security*, edited by Jim Blyth, Sven Dietrich, and L. Jean Camp, 7398:1–12. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. https://doi.org/10.1007/978-3-642-34638-5_1.

Croll, Grenville J. "BiEntropy - The Approximate Entropy of a Finite Binary String." *ArXiv:1305.0954 [Cs]*, September 15, 2013. http://arxiv.org/abs/1305.0954.

Glory, Farhana Zaman, Atif Ul Aftab, Olivier Tremblay-Savard, and Noman Mohammed. "Strong Password Generation Based On User Inputs." In *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 0416–23, 2019. https://doi.org/10.1109/IEMCON.2019.8936178.

Hinton, G.E., and R.R Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks | Science." Accessed May 19, 2021. https://science.sciencemag.org/content/313/5786/504.

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long Short-Term Memory." *Neural Computation* 9, no. 8 (November 15, 1997): 1735–80. https://doi.org/10.1162/neco.1997.9.8.1735.

Kingma, Diederik P., and Max Welling. "Auto-Encoding Variational Bayes." *ArXiv:1312.6114 [Cs, Stat]*, May 1, 2014. http://arxiv.org/abs/1312.6114.

Komanduri, Saranga, 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 2011 Annual Conference on Human Factors in Computing Systems - CHI '11*, 2595. Vancouver, BC, Canada: ACM Press, 2011. https://doi.org/10.1145/1978942.1979321.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." *ArXiv:1912.01703 [Cs, Stat]*, December 3, 2019. http://arxiv.org/abs/1912.01703.

Pennington, Jeffrey, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation." In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–43. Doha, Qatar: Association for Computational Linguistics, 2014. https://doi.org/10.3115/v1/D14-1162.

Princeton University. (2010). "About WordNet". Princeton University. Retrieved from: https://wordnet.princeton.edu/

Rehurek, Radim, and Petr Sojka. "Gensim–python framework for vector space modelling." *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic* 3, no. 2 (2011).

Shannon, C. E., & Weaver, W. (1949). “The mathematical theory of communication”. Urbana: University of Illinois Press

Yan, J., A. Blackwell, R. Anderson, and A. Grant. “Password Memorability and Security: Empirical Results.” *IEEE Security Privacy* 2, no. 5 (September 2004): 25–31. https://doi.org/10.1109/MSP.2004.81.

Yan, Jianxin, Alan Blackwell, Ross Anderson, and Alasdair Grant. “The Memorability and Security of Passwords – Some Empirical Results.” University of Cambridge, Computer Laboratory, 2000. https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-500.html?so=nx;htmle=1;info=.

Yang, Weining, Ninghui Li, Omar Chowdhury, Aiping Xiong, and Robert W. Proctor. “An Empirical Study of Mnemonic Sentence-Based Password Generation Strategies.” In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 1216–29. CCS ’16. New York, NY, USA: Association for Computing Machinery, 2016. https://doi.org/10.1145/2976749.2978346.