Adversarial Password Cracking

Suman Nepal, Isaac Kontomah, Ini Oguntola, Daniel Wang
Department of Electrical Engineering and Computer Science, MIT
Cambridge, MA
{nsuman, kontomah, ini, dwq}@mit.edu

***Abstract*—Password generation is a very common security issue. Users are faced with the problem of choosing good passwords and trying to make sure that the password they choose is as robust as possible. In this project, we train a generative adversarial network with a generator that tries to generate guesses to break a password scheme and a discriminator that tries to prevent the generator from generating a guess that breaks the password security scheme. We try to compare this scheme to Hashcat and also evaluate the strength of our adversarial system using zxcvbn, a low budget password estimator developed by Dropbox engineers. We also compare the effect of creating an extended dataset based on random permutations and language rules on the guessability of user passwords.**

I. Introduction

Generating passwords that are robust to attacks is still a major security concern. In an ideal situation, passwords are chosen completely randomly and are not easily cracked without compromising the underlying encryption scheme or some other weak point other than the passwords themselves. However, users tend to pick passwords that are easy for them to remember, which can drastically reduce the search space for a potential adversary.

Many password systems cracking systems take advantage of this human tendency via dictionary attacks, which attempt to match the hash of a given password from a list of commonly used passwords, with some success. More sophisticated systems can function based on a set of rules of language and grammar to expand the adversary search space beyond a fixed password list. However, these systems are still primarily based on a set of rules of language and grammar, so when a user sets a password outside of these rules it becomes harder to crack.

This paper explores password cracking with PassGAN [1]. The main idea behind PassGAN is to try to check the robustness of user passwords by using a generative adversarial network. The generator and discriminator act in a cat and mouse manner where the generator tries to generate fake passwords and make fool the discriminator to think they are real passwords. PassGAN replaces rule-based password guessing, and also password guessing based on data-driven Markov decision systems, with a adversarial methods using deep learning. This is done by training a neural network to determine password attributes autonomously, and using the knowledge of user password attributes to learn and mimic the distribution of previous passwords. The advantage that deep neural networks hold is that they can be trained without any a priori knowledge of any properties and structures of user password choices. This makes deep networks more efficient as compared to Markov models which implicitly assume that all relevant password characteristics can be defined in terms of $n-$ grams, and rule-based approaches which can guess only passwords that match with the available rules. As a result, samples generated using a neural network are not limited to a particular subset of the password space. Instead, neural networks can autonomously encode a wide range of password-guessing knowledge that includes and surpasses what is captured in human-generated rules and Markovian password generation processes.

In this work we reproduce the results from [1], as well as experimenting with alternate models of PassGAN trained on different or modified password lists. In addition, we analyze the strength of our models' generated passwords via zxcvbn [2]. Finally, we derive a probabilistic model for an alternative password strength checker based off of our trained PassGAN models.

II. Prior Work

Two widely used password guessing tools in use today are John the Ripper and HashCat, which expand on dictionary attacks by creating rules for password transformations. John the Ripper offers a standard dictionary attack, with password expansion rules based on behaviour in choosing passwords seen in user accounts from leaked databases. HashCat also offers a similar dictionary attack, but uses a distributed approach across GPUs for its implementation. In this work we primarily

use HashCat for comparison with neural-network based adversarial approaches.

Unfortunately, both John the Ripper and Hashcat exploit common password patterns that are hard-coded based on the developer's discretion, which includes analyzing the password structure. What if we use deep learning and have the machine learn the nuances and patterns from scratch, and perhaps pick up more sophisticated password patterns that humans don't notice?

One of said approaches is from Melicher et. al. [6], who used neural networks to model password strength, and were able to so effectiveness comparable to state-of-the-art. Another machine learning based approach to password guessing is PassGAN [1], which is the focus of this paper. PassGAN uses generative adversarial networks (GANs) to generate likely passwords, and achieves results comparable to standard password guessing tools such as John the Ripper and HashCat. Others have also obtained promising results using other deep learning approaches such as LSTMs [7] [8] [9].

III. PassGAN

A. Generative Adversarial Networks

PassGAN is an example of what are called *generative adversarial networks* (GANs) [3]. GANs are essentially an adversarial framework of multilayer perceptrons made up of a generator and discriminator. To learn the generators distribution p_g over data x, one defines a prior on input noise variables $p_z(z)$, then represent a mapping to data space as $G(z;\theta_g)$, where G is a differentiable function represented by a multilayer perceptron with parameters θ_g. One also defines a second multilayer perceptron $D(x;\theta_d)$ that outputs a single scalar. $D(x)$ represents the probability that x came from the data rather than p_g. One then trains D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize $log(1-G(z))$.

This can be though of as D and G playing a two-player minimax game with value function $V(G,D)$. We optimize the objective value function by solving the equation:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[log(D(x))] + \mathbb{E}_{z\sim p_z(z)}[log(1-G(z))] \quad (1)$$

B. How PassGAN Works

The generator network tries to generate samples from the same distribution as that of its training set $\mathcal{S} = \{x_1, x_2, ..., x_n\}$. Generative modeling relies on closed-form expressions that usually cannot capture the nuisance of real data. PassGAN trains a generative deep neural network G that takes as input a multi-dimensional random sample of passwords that is formed into a Gaussian or uniform distribution) to generate a sample from the desired distribution target password distribution. GANs transform the density estimation problem into a binary classification problem, in which the learning of the parameters of G is achieved by relying on a discriminator. The discriminator tries to prevent the generator from generator from generating a password distribution similar to the target password distribution while the discriminator tries to generate a fake password distribution and try to fool to discriminator to think it is a real distribution. They both end up playing the minimax game in equation (1) and optimize the value function $V(G,D)$ for the password distributions.

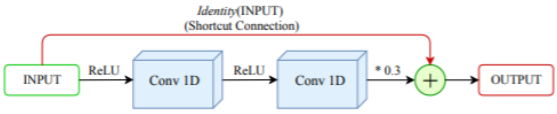


Fig. 1: Residual Block

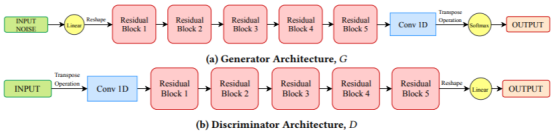


Fig. 2: PassGAN architecture

C. Architecture of PassGAN

The PassGAN is build up of several residual blocks each of which 1D convolutional blocks connected by identity short connection. The PassGAN generator has a reshape node followed by five residual nodes, one 1D convolutional node which outputs to the softmax node to generate probability distribution in the character set. The discriminator performs the same operations as the generator but in an opposite order as we can see in the Fig. 2. The generator will be initialized with random noise does not use training data set except for gradient update. The discriminator takes both fake and real inputs, processes the forward pass and computes gradient to optimize equation 1.

IV. Training the network

A. Sampling data

The RockYou data set contains a total of 31 million passwords among which 14 million are unique. We think the using the whole of the dataset is computationally intensive for this project and decided to uniformly sample 2.5 million passwords from the original 31 million passwords. This would ensure the distribution of the passwords will remain the same as the original distribution. We further restricted the length of the passwords to be less than or equal to 10 in order to make the training computationally feasible and ensure uniformity. The passwords with length less than 10 were padded.

We sampled additional 2.5 million passwords that were not included in the training samples in order to test the network output.

B. Training

The network was trained in a GPU accelerated machine for 70k iteration with batch size of 256. The training time was about 12 hours. The Jensen-Shannon divergence measure was plotted between the output distribution and real distribution as loss measure during training.

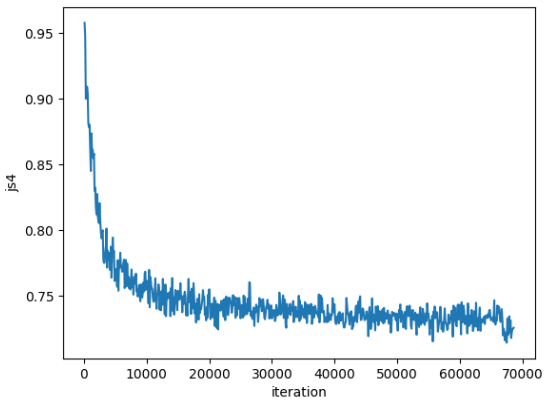


Fig. 3: Training Loss

C. Evaluation

For the evaluation of the trained network, we sampled 5 million passwords from the generator network and compared it's output the test data set we created above. About 274965 (5.5%) generated passwords were found the test set we chose earlier, and among them 63110 of them were unique.

Similarly we also tested generated password with the 32 million passwords leaked from the Ashley Madison website. This data set was not seen by the model at all. Again the accuracy rate was similar to the RockYou data set with 4.5% of the generated passwords present in the Ashley Madison passwords.

V. Accuracy of PassGAN between Initial Dataset and Random Permutation of Dataset

We found the the RockYou dataset included a significantly large amount of passwords that were essentially single words, like "password" or "football", so to expand that dataset into a more realistic set of passwords, ones that include slightly modified variations, such as "password1" or "p@s$w0rd". To address this concern, We wrote a Python program to randomly generate these variations.

The first issue that we need to address is the following: How can we accurately determine, through human observation, password variants for a simple existing password? One way would be to observe some more complex passwords via the RockYou dataset, and find certain patterns exhibited in a significant portion of these passwords. For example, we found that many users append a number to the end of their password, most commonly a single digit, or a 2/4 digit number, assumed to be a birthyear. Others substitute some words with l33t speak, which is substituting certain letters with symbols that resemble the replaced letter (like the "p@s$w0rd" example above). We enumerate these examples and observe the frequency of these patterns in the wild. That way, given enough samples, we can use the same probabilities to generate a whole new batch of altered passwords.

Our python program takes a common password as an input and n as the second input. From that, the function generates n slightly altered passwords from the original. If we run this function for every input password in the RockYou dataset, we can potentially get 20 times as much passwords for our new input dataset.

We did the training again in the similar fashion as the original dataset and got very similar results. The percentage accuracy only changed by 0.3 percentage points (upto 5.8%) on the RockYou and 0.2 (upto 4.7%) percentage points in the Ashley Madison dataset. This might be because of our constrained computational resources for training the big model for too long (generally about couple of days) and the enormous size of the data (possible to train in a PC only a sample of it). We then turn to analyze the strength of true passwords and the passwords generated by the GAN.

VI. Evaluating Password Strength with zxcvbn

Using the cracked passwords from training with PassGAN, we try to evaluate using zxcvbn, how hard these passwords passGAN cracked are to crack. zxcvbn [2] is a low-budget password strength estimator developed by Daniel Lowe Wheeler of Dropbox. Using leaked passwords, zxcvbn compares its estimations to four modern guessing attacks which are accurate and conservative at low magnitudes, suitable for mitigating online attacks. zxcvbn is a password strength estimator inspired by password crackers. Through pattern matching and conservative estimation, it can recognize and weighs 30000 common passwords, common names and surnames according to US census data, popular English words from Wikipedia and US television and movies, and other common language patterns and permutations such as dates, repeats (aaa), sequences $(abcd)$, keyboard patterns $(qwertyuiop)$, and $l33t$ speak. With zxcvbn, we get to know the estimated guesses required to crack a password, and the order of magnitude of the guesses required. The four main criterion zxcvbn uses which are based on the common password cracking schemes used:

- Online Throttling 100 per hour: An online attack on a service that rate limits password authorization attempts.
- Online no throttling 10 per second: An online attack on a service that does not rate limit.
- Offline Slow Hashing 10^4 per second: An offline attack which assumes multiple attackers with user-unique salting and a slow hash function. Uses a moderate work factor such as bcrypt, scrypt, PBKDF2.
- Offline Fast Hashing 10^{10} per second: An offline attack that uses a user-unique salting and a fast hash function like SHA-1, SHA-256 or MD5. Number of guesses per second ranges one billion to one trillion. Rate limiting means zxcvbn limits the amount of time an attacker can try to compute a given password before being denied access to trying to input any passwords.
  Cryptographic salting is the addition of random bits to each password instance before its hashing. Salts create unique passwords even in the instance of two users choosing the same passwords. Salts help in mitigate rainbow table attacks by forcing attackers to re-compute them using the salted hash functions.

A formalization of what an attacker knows is based on a heuristic search that follows the equation below:

$$argmin_{S \in \mathcal{S}} D^{|S|-1} + S! \prod_{m \in S} m \cdot guesses \quad (2)$$

Where $|S|$ is the length of the sequence, D is a constant.i If an attacker knows the pattern sequence with bounds on how many guesses needed for each pattern, the $\prod$ term denotes the number of guesses required in the worst case. The $|S|!$ term helps the guesser now know the number of patterns in the sequence but not the order. If the password contains a common word c, an uncommon word u, and a date d, there are $3!Z$ possible orderings to try: cud, ucd, duc,.The $D^{|S|1}$ term attempts to model a guesser who has no information on the length of the pattern sequence. Before attempting length$-|S|$ sequences, zxcvbn assumes that a guesser attempts lower-length pattern sequences first with a minimum of D guesses per pattern, trying a total of $\sum_{l=1}^{|S|-1} \approx D^{|S|-1}$ guesses for sufficiently large D.
The zxcvbn algorithm then returns a an integer strength bar from $0-4$ with the following rules:

- Too guessable: risky password, guesses $< 10^3$.
- Very guessable: protection from throttled online attacks, guesses $< 10^6$.
- Somewhat guessable: protection from unthrottled online attacks, guesses $< 10^8$.
- Safely unguessable: moderate protection from offline slow-hash scenario, guesses $< 10^{10}$.
- Very unguessable: strong protection from offline slow-hash scenario, guesses $\geq 10^{10}$.

After training on the passwords on the passGAN deep network, we then try to evaluate how hard on average the passwords cracked by passGAN are to guess using another independednt password evaluator, zxcvbn.

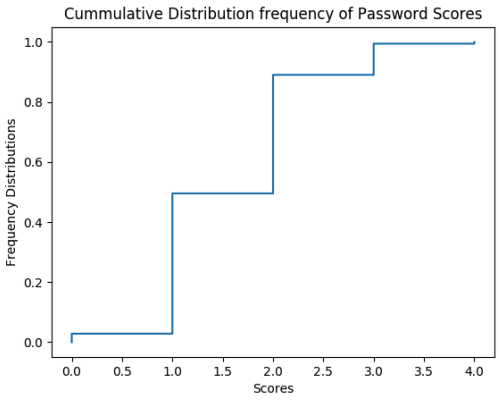
Below are the average scores, guesses and log guesses for the original RockYou dataset and the extended RockYou dataset.

Data	Avg. Score	Avg. Guesses	Log Guesses
RockYou	1.5925	5.32×10^6	6.2561
Ext RockYou	1.5561	6.8×10^7	6.166

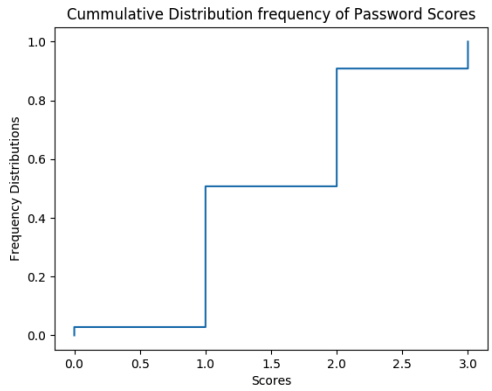
From the results, we see that the extended RockYou dataset in which we added more random variations and permutations to the passwords has a lower average score than the original RockYou dataset, we can then infer that the random permutations made it harder to crack passwords which makes because with more random

permutations that do not follow any rules of grammar or language models, it becomes harder to guess the passwords.
Below throttling and hashing parameters for the original RockYou dataset and the extended RockYou dataset.

Data	Throt/hr	Throt/Sec	FastHash	SlowHash
RockYou	5.3×10^9	1.9×10^{12}	5.3237	5.3×10^6
E. RockYou	6.8×10^6	2.5×10^9	0.0068	6.8×10^3

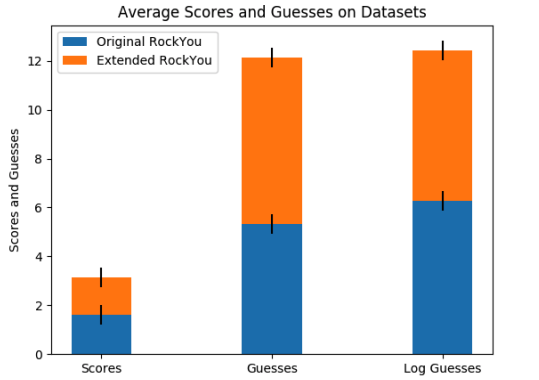


CDF of Original RockYou

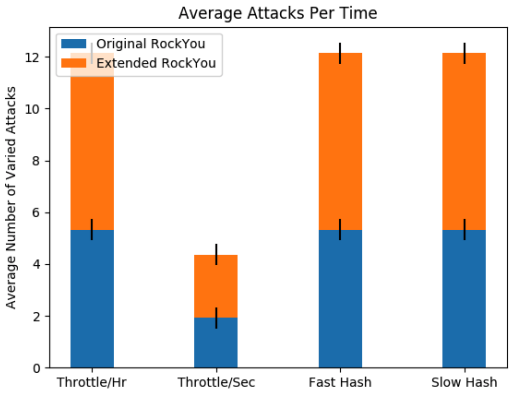


CDF on extended RockYou

Fig. 4: *CDF of Scores on RockYou and Extended RockYou*

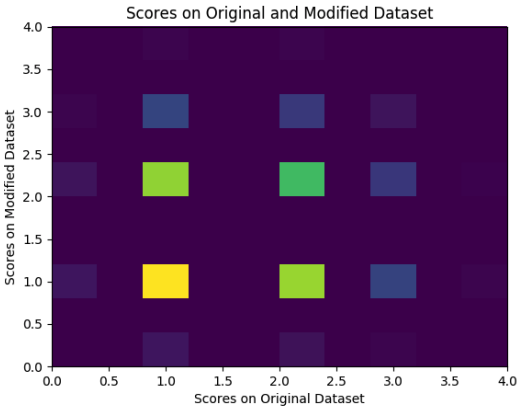


Average Scores on guesses on datasets



Average Attacks on datasets

Fig. 5: *Average Scores,guesses and attacks on datasets*



Scores on datasets

Guesses on datasets

Fig. 6: *Scores and guesses on datasets*

From the results for the original RockYou and extended RockYou datasets, we see that the average score for the RockYou dataset is lesser than the average score for the extended RockYou dataset, this follows what we expect because we incorporate a lot of intricacies into the extended RockYou dataset which should make it a bit more harder for attackers to guesses the passwords rightly because the passwords get more and more random with more permutations being added. Logically, that means we should also expect the number of guesses on average that attackers need to make on the extended RockYou data set to be larger than the average number of guesses needed on the original RockYou dataset. Which the results from the zxcvbn confirm.

VII. Password Strength Checker with PassGAN

Like the zxcvbn, we can similarly use PassGAN for analyzing how strong a given password is. We can define the strength of password as how many passwords the Passgan generates before it outputs the desired password. A simple measure of strength of password will be how long it takes for the PassGAN to guess the password. It will be computationally expensive and impractical to run PassGAN until it outputs password (if it ever does). But we can use a probabilistic model in order to guess that number. The probability to guess the certain password by the PassGAN is the product of probabilities of each character from the conditional posterior distribution. The softmax output of PassGAN acts as posterior distribution over character set. We assign the difficulty score to the password -log of base 10.

Algorithm 1: Get prediciton probability

```
Result: score
Input : password, model, charmap,;
prob = 1;
for char in password do
    char_prob = model.getProbability(char);
    prob = prob × char_prob
end
score = -log(prob)
```

From the zxcvbn library, we can get benchmark of how many guesses (output number) it takes to brute force a password. We checked our score with log of zxcvbn output number. The correlation coefficient for our scores and the zxcvbn output number for the first 5000 frequent passwords was 0.26 and for 63000 matched password was 0.17.

Acknowledgments

We would like to thank the 6.857 staff at MIT for their instrumental guidance and feedback throughout this project.

References

[1] B. Hitaj, P. Gasti, G. Ateniese, and F. Perez-Cruz, "PassGAN: A Deep Learning Approach for Password Guessing," arXiv:1709.00440v3 [cs.CR], Feb 2019.

[2] D. L. Wheeler, zxcvbn: Low-Budget Password Strength Estimation, Proceedings of the 25th USENIX Security Symposium, Aug 2016.

[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks,". arXiv:1406.2661v1. Jun 2014.

[4] M. Arjovsky, S. Chintala, and L. Bottou. "Wasserstein GAN", arXiv:1701.07875v3 [stat.ML], Dec 2017;

[5] L. Castro, H. Lang, S. Liu, C. Mata. Modeling Password Guessing with Neural Networks, https://courses.csail.mit.edu/6.857/2017/project/13.pdf, May 2017.

[6] W. Melicher, B. Ur, S. M. Segreti, S. Komanduri, L. Bauer, N. Christin, and L. Faith Cranor, "Fast, Lean, Accurate: Modeling Password Guessability Using Neural Networks", 2016.

[7] Y. Liu, Z. Xia, P. Yi, Y. Yao, T. Xie, W. Wang, and T. Zhu, GENPass: A General Deep Learning Model for Password Guessing with PCFG Rules and Adversarial Generation, IEEE, 2018.

[8] C. Olsen, A Machine Learning Approach to Predicting Passwords, 2018.

[9] I. Xu, C. Ge, W. Qiu, Z. Huang, Z. Gong, J. Guo, and Huijuan Lian. Password guessing based on Lstm Recurrent Neural Networks, 2017.