PassGAN: A Deep Learning Approach for Password Guessing* **

Briland Hitaj[1], Paolo Gasti[2], Giuseppe Ateniese[1], and Fernando Perez-Cruz[1,3]

[1] Stevens Institute of Technology, Hoboken, NJ 07030, USA
{bhitaj,gatenies}@stevens.edu
[2] New York Institute of Technology, New York, NY 10023, USA
pgasti@nyit.edu
[3] Swiss Data Science Center, (ETH Zurich and EPFL)
fernando.perezcruz@sdsc.ethz.ch

Abstract. State-of-the-art password guessing tools, such as HashCat and John the Ripper, enable users to check billions of passwords per second against password hashes. In addition to performing straightforward dictionary attacks, these tools can expand password dictionaries using password generation rules, such as concatenation of words (e.g., "password123456") and *leet speak* (e.g., "password" becomes "p4s5w0rd"). Although these rules work well in practice, creating and expanding them to model further passwords is a labor-intensive task that requires specialized expertise.

To address this issue, in this paper we introduce PassGAN, a novel approach that replaces human-generated password rules with theory-grounded machine learning algorithms. Instead of relying on manual password analysis, PassGAN uses a Generative Adversarial Network (GAN) to autonomously learn the distribution of real passwords from actual password leaks, and to generate high-quality password guesses. Our experiments show that this approach is very promising. When we evaluated PassGAN on two large password datasets, we were able to surpass rule-based and state-of-the-art machine learning password guessing tools. However, in contrast with the other tools, PassGAN achieved this result without any a-priori knowledge on passwords or common password structures. Additionally, when we combined the output of PassGAN with the output of HashCat, we were able to match 51%-73% more passwords than with HashCat alone. This is remarkable, because it shows that PassGAN can autonomously extract a considerable number of password properties that current state-of-the art rules do not encode.

Keywords: Passwords · Privacy · Generative Adversarial Networks (GAN) · Deep Learning

* Due to space limitations, an extended version of the paper can be found here: https://arxiv.org/abs/1709.00440

** A preliminary version of this paper appeared in NeurIPS 2018 Workshop on Security and Machine Learning (SecML'18) [24].

1 Introduction

Passwords are the most popular authentication method, mainly because they are easy to implement, require no special hardware or software, and are familiar to users and developers [27]. Unfortunately, multiple password database leaks have shown that users tend to choose easy-to-guess passwords [10,14,36], primarily composed of common strings (e.g., `password`, `123456`, `iloveyou`), and variants thereof.

Password guessing tools provide a valuable tool for identifying weak passwords when they are stored in hashed form [49,53]. The effectiveness of password guessing software relies on the ability to quickly test a large number of highly likely passwords against each password hash. Instead of exhaustively trying all possible character combinations, password guessing tools use words from dictionaries and previous password leaks as candidate passwords. State-of-the-art password guessing tools, such as John the Ripper [55] and HashCat [22], take this approach one step further by defining heuristics for password transformations, which include combinations of multiple words (e.g., `iloveyou123456`), mixed letter case (e.g., `iLoVeyOu`), and *leet speak* (e.g., `il0v3you`). These heuristics, in conjunction with Markov models, allow John the Ripper and HashCat to generate a large number of *new* highly likely passwords.

While these heuristics are reasonably successful in practice, they are ad-hoc and based on intuitions on how users choose passwords, rather than being constructed from a principled analysis of large password datasets. For this reason, each technique is ultimately limited to capturing a specific subset of the password space which depends upon the intuition behind that technique. Further, developing and testing new rules and heuristics is a time-consuming task that requires specialized expertise, and therefore has limited scalability.

1.1 Our Approach

To address these shortcomings, in this paper we propose to replace rule-based password guessing, as well as password guessing based on simple data-driven techniques such as Markov models, with a novel approach based on deep learning. At its core, our idea is to train a neural network to determine autonomously password characteristics and structures, and to leverage this knowledge to generate new samples that follow the same distribution. We hypothesize that deep neural networks are expressive enough to capture a large variety of properties and structures that describe the majority of user-chosen passwords; at the same time, neural networks can be trained without any a-priori knowledge or an assumption of such properties and structures. This is in stark contrast with current approaches such as 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.

To test this hypothesis, in this paper we introduce PassGAN, a new approach for generating password guesses based on deep learning and Generative Adversarial Networks (GANs) [18]. GANs are recently-introduced machine learning tools designed to perform density estimation in high-dimensional spaces [18]. GANs perform implicit generative modeling by training a deep neural network architecture that is fed a simple random distribution (e.g., Gaussian or uniform) and by generating samples that follow the distribution of the available data. In a way, they implicitly model the inverse of the cumulative distribution with a deep neural network, i.e., $\mathbf{x} = F_\theta^{-1}(s)$ where s is a uniformly distributed random variable. To learn the generative model, GANs use a cat-and-mouse game, in which a deep generative network (G) tries to mimic the underlying distribution of the samples, while a discriminative deep neural network (D) tries to distinguish between the original training samples (i.e., "true samples") and the samples generated by G (i.e., "fake samples"). This adversarial procedure forces D to leak the relevant information about the training data. This information helps G to adequately reproduce the original data distribution.

PassGAN leverages this technique to generate new password guesses. We train D using a list of leaked passwords (real samples). At each iteration, the output of PassGAN (fake samples) gets closer to the distribution of passwords in the original leak, and therefore more likely to match real users' passwords. To the best of our knowledge, this work is the first to use GANs for this purpose.

1.2 Contributions

PassGAN represents a principled and theory-grounded take on the generation of password guesses. We explore and evaluate different neural network configurations, parameters, and training procedures, to identify the appropriate balance between *learning* and *overfitting*, and report our results. Specifically, our contributions are as follows:

1. We show that a GAN can generate high-quality password guesses. Our GAN is trained on a portion of the RockYou dataset [57], and tested on two different datasets: (1) another (distinct) subset of the RockYou dataset; and (2) a dataset of leaked passwords from LinkedIn [35]. In our experiments, we were able to match 1,350,178 (43.6%) *unique* passwords out of 3,094,199 passwords from the RockYou dataset, and 10,478,322 (24.2%) unique passwords out of 43,354,871 passwords from the LinkedIn dataset. To quantify the ability of PassGAN to generate new passwords, we removed from the testing set all passwords that were present also in the training set. This resulted in testing sets of size 1,978,367 and 40,593,536 for RockYou and LinkedIn, respectively. In this setting, PassGAN was able to match 676,439 (34.6%) samples in the RockYou testing set and 8,878,284 (34.2%) samples in the LinkedIn set. Moreover, the overwhelming majority of passwords generated

by PassGAN that did not match the testing sets still "looked like" human-generated passwords, and thus could potentially match real user accounts not considered in our experiments.
2. We show that PassGAN is competitive with state-of-the-art password generation rules. Even though these rules were specially tuned for the datasets used in our evaluation, the quality of PassGAN's output was comparable to that of password rules.
3. With password generation rules, the number of unique passwords that can be generated is defined by the number of rules and by the size of the password dataset used to instantiate them. In contrast, PassGAN can output a practically unbounded number of password guesses. Crucially, our experiments show that with PassGAN the number of matches increases steadily with the number of passwords generated, Table 1. This is important because it shows that the output of PassGAN is not restricted to a small subset of the password space.
4. PassGAN is competitive with current state of the art password guessing algorithms based on deep neural networks [38], matching the performance of Melicher et al. [38], (indicated as FLA in the rest of the paper).
5. We show that PassGAN can be effectively used to *augment* password generation rules. In our experiments, PassGAN matched passwords that were not generated by any password rule. When we combined the output of PassGAN with the output of HashCat, we were able to guess between 51% (case of RockYou) and 73% (case of LinkedIn) additional unique passwords compared to HashCat alone.

We consider this work as the first step toward a fully automated generation of high-quality password guesses. We argue that this work is relevant, important, and timely. *Relevant*, because despite numerous alternatives [50,63,16,13,71], we see little evidence that passwords will be replaced any time soon. *Important*, because establishing the limits of password guessing—and better understanding how guessable real-world passwords are—will help make password-based systems more secure. And *timely*, because recent leaks containing hundreds of millions of passwords [15] provide a formidable source of data for attackers to compromise systems, and for system administrators to re-evaluate password policies.

1.3 Organization

The rest of this paper is organized as follows. In Section 2, we briefly overview GANs, and password guessing, and provide a summary of the relevant state of the art. Section 3 provides details regarding our experimental setup, architectural and training choices for PassGAN, and the hyperparameters used in our evaluation. We report on the evaluation of PassGAN, and on the comparison with state-of-the-art password guessing techniques, in Section 4. We summarize our findings and discuss their implications, in Section 5. We conclude in Section 6.

2 Background and Related Work

2.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs) translate the current advances in deep neural networks for discriminative machine learning to (implicit) generative modeling. The goal of GANs is to generate samples from the same distribution as that of its training set $\mathcal{S} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$. Generative modeling [45] typically relies on closed-form expressions that, in many cases, cannot capture the nuisance of real data. GANs train a generative deep neural network G that takes as input a multi-dimensional random sample $\mathbf{z}$ (from a Gaussian or uniform distribution) to generate a sample from the desired 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 discriminative deep neural network D that needs to distinguish between the "true" samples in $\mathcal{S}$ and the "fake" samples produced by G. More formally, the optimization problem solved by GANs can be summarized as follows:

$$\min_{\theta_G} \max_{\theta_D} \sum_{i=1}^{n} \log f(\mathbf{x}_i; \theta_D) + \sum_{j=1}^{n} \log(1 - f(g(\mathbf{z}_j; \theta_G); \theta_D)), \tag{1}$$

where $f(\mathbf{x}; \theta_D)$ and $g(\mathbf{z}_j; \theta_G)$, respectively, represent D and G. The optimization shows the clash between the goals of the discriminator and generator deep neural networks. Since the original work by Goodfellow et al. [18], there have been several improvements on GANs [48,2,54,62,20,34,46,58,43,3,29,39,7,11,70,66,51,26] [5,42,4,9,25,40,72], where each new paper provides novel improvements in the domain. In this paper, we rely on IWGAN [20] as a building foundation for PassGAN, being that IWGAN [20] is among the first, most stable approaches for text generation via GANs.

2.2 Password Guessing

Password guessing attacks are probably as old as password themselves [44], with more formal studies dating back to 1979 [41]. In a password guessing attack, the adversary attempts to identify the password of one or more users by repeatedly testing multiple candidate passwords.

Two popular modern password guessing tools are John the Ripper (JTR) [55] and HashCat [22]. Both tools implement multiple types of password guessing strategies, including: exhaustive brute-force attacks; dictionary-based attacks; rule-based attacks, which consist in generating password guesses from transformations of dictionary words [60,59]; and Markov-model-based attacks [56,37]. JTR and HashCat are notably effective at guessing passwords. Specifically, there have been several instances in which well over 90% of the passwords leaked from online services have been successfully recovered [52].

Markov models were first used to generate password guesses by Narayanan et al. [47]. Their approach uses manually defined password rules, such as which

portion of the generated passwords is composed of letters and numbers. Weir et al. [68] subsequently improved this technique with Probabilistic Context-Free Grammars (PCFGs). With PCFGs, Weir et al. [68] demonstrated how to "learn" these rules from password distributions. Ma et al. [36] and Durmuth et al. [14] have subsequently extended this early work.

To the best of our knowledge, the first work in the domain of passwords utilizing neural networks dates back to 2006 by Ciaramella et al. [8]. Recently, Melicher et al. [38] introduced FLA, a password guessing method based on recurrent neural networks [19,64]. However, the primary goal of these works consists in providing means for password strength estimation. For instance, Melicher et al. [38] aim at providing fast and accurate password strength estimation (thus FLA acronym), while keeping the model as lightweight as possible, and minimizing accuracy loss. By keeping the model lightweight, FLA instantiates a password strength estimator that can be used in browsers through a (local) JavaScript implementation. To achieve this goal, FLA uses weight clipping without significantly sacrificing accuracy. In contrast, PassGAN focuses on the task of password guessing and attempts to do so with no a priori knowledge or assumption on the Markovian structure of user-chosen passwords.

3 Experiment Setup

To leverage the ability of GANs to estimate the probability effectively distribution of passwords from the training set, we experimented with a variety of parameters. In this section, we report our choices on specific GAN architecture and hyperparameters.

We instantiated PassGAN using the *Improved training of Wasserstein GANs* (IWGAN) of Gulrajani et al. [20]. The IWGAN implementation used in this paper relies on the ADAM optimizer [30] to minimize the training error.

The following hyper-parameters characterize our model:

- **Batch size**, which represents the number of passwords from the training set that propagate through the GAN at each step of the optimizer. We instantiated our model with a batch size of 64.
- **Number of iterations**, which indicates how many times the GAN invokes its forward step and its back-propagation step [61,32,33]. In each iteration, the GAN runs one generator iteration and one or more discriminator iterations. We trained the GAN using various number of iterations and eventually settled for 199,000 iterations, as further iterations provided diminishing returns in the number of matches.
- **Number of discriminator iterations per generator iteration**, which indicates how many iterations the discriminator performs in each GAN iteration. The number of discriminator iterations per generative iteration was set to 10, which is the default value used by IWGAN.
- **Model dimensionality**, which represents the number of dimensions for each convolutional layer. We experimented using 5 residual layers for both

the generator and the discriminator, with each of the layers in both deep neural networks having 128 dimensions.
- **Gradient penalty coefficient** (λ), which specifies the penalty applied to the norm of the gradient of the discriminator with respect to its input [20]. Increasing this parameter leads to a more stable training of the GAN [20]. In our experiments, we set the value of the gradient penalty to 10.
- **Output sequence length**, which indicates the maximum length of the strings generated by the generator (G). We modified the length of the sequence generated by the GAN from 32 characters (default length for IWGAN) to 10 characters, to match the maximum length of passwords used during training. We padded passwords shorter than 10 characters using accent symbol (i.e., "`"); we then removed it from the output of PassGAN.
- **Size of the input noise vector (seed)**, which determines how many random numbers from a normal distribution are fed as input to G to generate samples. We set this size to 128 floating point numbers.
- **Maximum number of examples**, which represents the maximum number of training items (passwords, in the case of PassGAN) to load. The maximum number of examples loaded by the GAN was set to the size of the entire training dataset.
- **Adam optimizer's hyper-parameters**:
  - **Learning rate**, i.e., how quickly the weights of the model are adjusted
  - **Coefficient** β_1, which specifies the decaying rate of the running average of the gradient.
  - **Coefficient** β_2, which indicates the decaying rate of the running average of the square of the gradient.

  Coefficients β_1 and β_2 of the Adam optimizer were set to 0.5 and 0.9, respectively, while the learning rate was 10^{-4}. These parameters are the default values used by Gulrajani et al. [20].

Our experiments were run using the TensorFlow implementation of IWGAN found at [21]. We used TensorFlow version 1.2.1 for GPUs [1], with Python version 2.7.12. All experiments were performed on a workstation running Ubuntu 16.04.2 LTS, with 64GB of RAM, a 12-core 2.0 GHz Intel Xeon CPU, and an NVIDIA GeForce GTX 1080 Ti GPU with 11GB of global memory.

3.1 Password Sampling Procedure for HashCat, JTR, Markov Model, PCFG and FLA

We used the portion of RockYou dataset selected for training, see Section 4.1, as the input dataset to HashCat Best64, HashCat gen2, JTR Spiderlab rules, Markov Model, PCFG, and FLA, and generated passwords as follows:

- We instantiated HashCat and JTR's rules using passwords from the training set sorted by frequency in descending order (as in [38]). HashCat Best64 generated 754,315,842 passwords, out of which 361,728,683 were unique and of length 10 characters or less. Note that this was the maximum number of

samples produced by Best64 rule-set for the given input set, i.e., RockYou training set. With HashCat gen2 and JTR SpiderLab we uniformly sampled a random subset of size 10^9 from their output. This subset was composed of passwords of length 10 characters or less.
- For FLA, we set up the code from [31] according to the instruction provided in [17]. We trained a model containing 2-hidden layers and 1 dense layer of size 512. We did not perform any transformation (e.g., removing symbols, or transforming all characters to lowercase) on the training set for the sake of consistency with the other tools. Once trained, FLA enumerates a subset of its output space defined by a probability threshold p: a password belongs to FLA's output if and only if its estimated probability is at least p. In our experiments, we set $p = 10^{-10}$. This resulted in a total of 747,542,984 passwords of length 10 characters or less. Before using these passwords in our evaluation, we sorted them by probability in descending order.
- We generated 494,369,794 unique passwords of length 10 or less using the 3-gram Markov model. We ran this model using its standard configuration [12].
- We generated 10^9 unique passwords of length 10 or less using the PCFG implementation of Weir et al. [67].

4 Evaluation

4.1 Training and Testing

To evaluate the performance of PassGAN, and to compare it with state-of-the-art password generation rules, we first trained the GAN, JTR, HashCat, the Markov model, PCFG, and FLA on a large set of passwords from the RockYou password leak [57].[4] Entries in this dataset represent a mixture of common and complex passwords.

RockYou Dataset The RockYou dataset [57] contains 32,503,388 passwords. We selected all passwords of length 10 characters or less (29,599,680 passwords, which correspond to 90.8% of the dataset), and used 80% of them (23,679,744 total passwords, 9,926,278 unique passwords) to train each password guessing tool. We refer the reader to Section 3.1 for further details on the training procedure of each tool. For testing, we computed the (set) difference between the remaining 20% of the dataset (5,919,936 total passwords, 3,094,199 unique passwords) and the training test. The resulting 1,978,367 entries correspond to passwords that were not previously observed by the password guessing tools. This allowed us to count only non-trivial matches in the testing set.

LinkedIn Dataset We also tested each tool on passwords from the LinkedIn dataset [35], of length up to 10 characters, and that were not present in the training set. The LinkedIn dataset consists of 60,065,486 total unique passwords

[4] We consider the use of publicly available password datasets to be ethical, and consistent with security research best practices (see, e.g., [10,38,6]).

(43,354,871 unique passwords with length 10 characters or less), out of which 40,593,536 were not in the training dataset from RockYou. (Frequency counts were not available for the LinkedIn dataset.) Passwords in the LinkedIn dataset were exfiltrated as hashes, rather than in plaintext. As such, the LinkedIn dataset contains only plaintext passwords that tools such as JTR and HashCat were able to recover, thus giving rule-based systems a potential edge.

Our training and testing procedures showed: (1) how well PassGAN predicts passwords when trained and tested on the same password distribution (i.e., when using the RockYou dataset for both training and testing); and (2) whether PassGAN generalizes across password datasets, i.e., how it performs when trained on the RockYou dataset, and tested on the LinkedIn dataset.

4.2 PassGAN's Output Space

To evaluate the size of the password space generated by PassGAN, we generated several password sets of sizes between 10^4 and 10^{10}. Our experiments show that, as the number of passwords increased, so did the number of unique (and therefore new) passwords generated. Results of this evaluation are reported in Table 1.

Table 1: Number of passwords generated by PassGAN that match passwords in the RockYou testing set. Results are shown in terms of unique matches.

Passwords Generated	**Unique Passwords**	**Passwords matched in testing set, and not in training set (1,978,367 unique samples)**
10^4	9,738	103 (0.005%)
10^5	94,400	957 (0.048%)
10^6	855,972	7,543 (0.381%)
10^7	7,064,483	40,320 (2.038%)
10^8	52,815,412	133,061 (6.726%)
10^9	356,216,832	298,608 (15.094%)
10^{10}	2,152,819,961	515,079 (26.036%)
$2 \cdot 10^{10}$	3,617,982,306	584,466 (29.543%)
$3 \cdot 10^{10}$	4,877,585,915	625,245 (31.604%)
$4 \cdot 10^{10}$	6,015,716,395	653,978 (33.056%)
$5 \cdot 10^{10}$	7,069,285,569	676,439 (34.192%)

When we increased the number of passwords generated by PassGAN, the rate at which new unique passwords were generated decreased only slightly. Similarly, the rate of increase of the number of matches (shown in Table 1) diminished slightly as the number of passwords generated increased. This is to be expected, as the simpler passwords are matched early on, and the remaining (more complex) passwords require a substantially larger number of attempts in order to be matched.

Impact of Training Process on Overfitting. Training a GAN is an iterative process that consists of a large number of iterations. As the number of iterations increases, the GAN learns more information from the distribution of the data.

Fig. 1: Number of unique passwords generated by PassGAN on various checkpoints, matching the RockYou testing set. The x axis represents the number of iterations (checkpoints) of PassGAN's training process. For each checkpoint, we sampled 10^8 passwords from PassGAN.

However, increasing the number of steps also increases the probability of overfitting [18,69].

To evaluate this tradeoff on password data, we stored intermediate training checkpoints and generated 10^8 passwords at each checkpoint. Figure 1 shows how many of these passwords match with the content of the RockYou testing set. In general, the number of matches increases with the number of iterations. This increase tapers off around 125,000-135,000 iterations, and then again around 190,000-195,000 iterations, where we stopped training the GAN. This indicates that further increasing the number of iterations will likely lead to overfitting, thus reducing the ability of the GAN to generate a wide variety of highly likely passwords. Therefore, we consider this range of iterations adequate for the RockYou training set.

4.3 Evaluating the Passwords Generated by PassGAN

To evaluate the quality of the output of PassGAN, we generated $5 \cdot 10^{10}$ passwords, out of which roughly $7 \cdot 10^9$ were unique. We compared these passwords with the outputs of length 10 characters or less from HashCat Best64, HashCat gen2, JTR SpiderLab, FLA, PCFG, and Markov model, see Section 3.1 for the configuration and sampling procedures followed for each of these tools.

In our comparisons, we aimed at establishing whether PassGAN was able to meet the performance of the other tools, *despite its lack of any a-priori knowledge on password structures*. This is because we are primarily interested in determining whether the properties that PassGAN autonomously extracts from a list of passwords can represent enough information to compete with state-of-the-art human-generated rules and Markovian password generation processes.

Table 2: Number of matches generated by each password guessing tool against the RockYou testing set, and corresponding number of password generated by PassGAN to outperform each tool. Matches for HashCat Best64 and FLA were obtained by exhaustively enumerating the entire output of each tool. The minimum probability threshold for FLA was set to $p = 10^{-10}$.

Approach	(1) Unique Passwords	(2) Matches	(3) Number of passwords required for PassGAN to outperform (2)	(4) PassGAN Matches
JTR Spyderlab	10^9	461,395 (23.32%)	$1.4 \cdot 10^9$	461,398 (23.32%)
Markov Model 3-gram	$4.9 \cdot 10^8$	532,961 (26.93%)	$2.47 \cdot 10^9$	532,962 (26.93%)
HashCat gen2	10^9	597,899 (30.22%)	$4.8 \cdot 10^9$	625,245 (31.60%)
HashCat Best64	$3.6 \cdot 10^8$	630,068 (31.84%)	$5.06 \cdot 10^9$	630,335 (31.86%)
PCFG	10^9	486,416 (24.59%)	$2.1 \cdot 10^9$	511,453 (25.85%)
FLA $p = 10^{-10}$	$7.4 \cdot 10^8$	652,585 (32.99%)	$6 \cdot 10^9$	653,978 (33.06%)

Our results show that, for each of the tools, PassGAN was able to generate at least the same number of matches. Additionally, to achieve this result, PassGAN needed to generate a number of passwords that was within one order of magnitude of each of the other tools. This holds for both the RockYou and the LinkedIn testing sets. This is not unexpected, because while other tools rely on prior knowledge on passwords for guessing, PassGAN does not. Table 2 summarizes our findings for the RockYou testing set, while Table 3 shows our results for the LinkedIn test set.

Our results also show that PassGAN has an advantage with respect to rule-based password matching when guessing passwords from a dataset different from the one it was trained on. In particular, PassGAN was able to match more passwords than HashCat within a smaller number of attempts ($2.1 \cdot 10^9$ – $3.6 \cdot 10^9$ for LinkedIn, compared to $4.8 \cdot 10^9$ – $5.06 \cdot 10^9$ for RockYou).

4.4 Combining PassGAN with HashCat

To maximize the number of passwords guessed, the adversary would typically use the output of multiple tools in order to combine the benefits of rule-based tools (e.g., fast password generation) and ML-based tools (e.g., generation of a large number of guesses).

To evaluate PassGAN in this setting, we removed all passwords matched by HashCat Best64 (the best performing set of rules in our experiments) from the RockYou and LinkedIn testing sets. This led to two new test sets, containing 1,348,300 (RockYou) and 33,394,178 (LinkedIn) passwords, respectively.

Our results show that the number of matches steadily increases with the number of samples produced by PassGAN. In particular, when we used $7 \cdot 10^9$ passwords from PassGAN, we were able to match 51% (320,365) of passwords from the "new" RockYou dataset, and 73% (5,262,427) additional passwords

Table 3: Number of matches generated by each password guessing tool against the LinkedIn testing set, and corresponding number of password generated by PassGAN to outperform each tool. Matches for HashCat Best64 and FLA were obtained by exhaustively enumerating the entire output of each tool. The minimum probability threshold for FLA was set to $p = 10^{-10}$.

Approach	(1) Unique Passwords	(2) Matches	(3) Number of passwords required for PassGAN to outperform (2)	(4) PassGAN Matches
JTR Spyderlab	10^9	6,840,797 (16.85%)	$2.7 \cdot 10^9$	6,841,217 (16.85%)
Markov Model 3-gram	$4.9 \cdot 10^8$	5,829,786 (14.36%)	$1.6 \cdot 10^9$	5,829,916 (14.36%)
HashCat gen2	10^9	6,308,515 (15.54%)	$2.1 \cdot 10^9$	6,309,799 (15.54%)
HashCat Best64	$3.6 \cdot 10^8$	7,174,990 (17.67%)	$3.6 \cdot 10^9$	7,419,248 (18.27%)
PCFG	10^9	7,288,553 (17.95%)	$3.6 \cdot 10^9$	7,419,248 (18.27%)
FLA $p = 10^{-10}$	$7.4 \cdot 10^8$	8,290,173 (20.42%)	$6 \cdot 10^9$	8,519,060 (21.00%)

from the "new" LinkedIn dataset. This confirms that combining rules with machine learning password guessing is an effective strategy. Moreover, it confirms that PassGAN can capture portions of the password space not covered by rule-based approaches. With this in mind, a recent version of HashCat [23] introduced a generic password candidate interface called "slow candidates", enabling the use of tools such as PCFGs [68], OMEN [14], PassGAN, and more with HashCat.

4.5 Comparing PassGAN with FLA

In this section, we concentrate on comparing PassGAN with FLA having a particular focus on the probability estimation. FLA is based on recurrent neural networks [19,64], and typically the model is trained on password leaks from several websites, in our case the RockYou training set. During password generation, the neural network generates one password character at a time. Each new character (including a special end-of-password character) is computed based on its probability, given the current output state, in what is essentially a Markov process. Given a trained FLA model, FLA outputs the following six fields: 1. password, 2. the probability of that password, 3. the estimated output guess number, i.e., the strength of that password, 4. the standard deviation of the randomized trial for this password (in units of the number of guesses), 5. the number of measurements for this password and 6. the estimated confidence interval for the guess number (in units of the number of guesses). The evaluation presented in [38] shows that their technique outperforms Markov models, PCFGs and password composition rules commonly used with JTR and HashCat, when testing a large number of password guesses (in the 10^{10} to 10^{25} range).

We believe that one of the limitations of FLA resides precisely in the Markovian nature of the process used to estimate passwords. For instance, `123456`; `12345`; and, `123456789` are the three most common passwords in the RockYou

dataset, being roughly one every 66-passwords. Similarly, the most common passwords produced by FLA start with "`123`" or use the word "`love`". In contrast, PassGAN's most commonly generated passwords, tend to show more variability with samples composed of names, the combination of names and numbers, and more. When compared with the RockYou training set, the most likely samples from PassGAN exhibit closer resemblance to the training set and its probabilities than FLA does. We argue that due to the Markovian structure of the password generation process in FLA, any password characteristic that is not captured within the scope of an $n-$gram, might not be encoded by FLA. For instance, if a meaningful subset of 10-character passwords is constructed as the concatenation of two words (e.g., `MusicMusic`), any Markov process with $n \leq 5$ will not be able to capture this behavior properly. On the other hand, given enough examples, the neural network used in PassGAN will be able to learn this property. As a result, while password `pookypooky` was assigned a probability $p \approx 10^{-33}$ by FLA (with an estimated number of guessing attempts of about 10^{29}), it was guessed after roughly 10^8 attempts by PassGAN.

To investigate further on the differences between PassGAN and FLA, we computed the number of passwords in the RockYou testing set for which FLA required at least 10^{10} attempts and that PassGAN was able to guess within its first $7 \cdot 10^9$ samples. These are the passwords to which FLA assigns low probabilities, despite being chosen by some users. Because PassGAN can model them, we conclude that the probabilities assigned by FLA to these passwords are incorrect. Figure 2 presents our result as the ratio between the passwords matched by FLA at a particular number of guessing attempts, and by PassGAN within its first $7 \cdot 10^9$ attempts. Our results show that PassGAN can model a number of passwords more correctly than FLA. However, this advantage decreased as the number of attempts required for FLA to guess a password increased, i.e., as the estimated probability of that password decreased. This shows that, in general, the two tools agree on assigning probabilities to passwords.

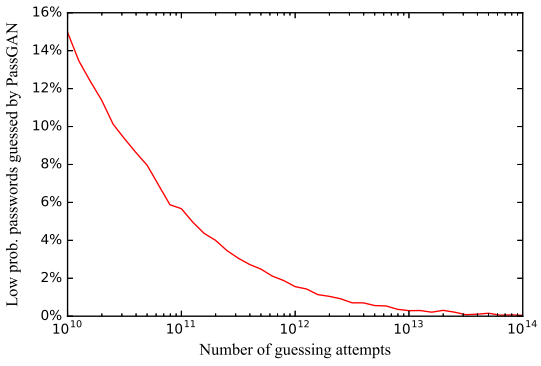

Fig. 2: Percentage of passwords matched by FLA at a particular number of guesses, that are matched by PassGAN in at most $7 \cdot 10^9$ attempts.

Table 4: Sample of passwords generated by PassGAN that did not match the testing sets.

love42743	ilovey2b93	paolo9630	italyit
sadgross	usa2598	s13trumpy	trumpart3
ttybaby5	dark1106	vamperiosa	~dracula
saddracula	luvengland	albania.	bananabake
paleyoung	@crepess	emily1015	enemy20
goku476	coolarse18	iscoolin	serious003
nyc1234	thepotus12	greatrun	babybad528
santazone	apple8487	1loveyoung	bitchin706
toshibaod	tweet1997b	103tears	1holys01

4.6 A Closer Look at Non-matched Passwords

We inspected a list of passwords generated by PassGAN that did not match any of the testing sets and determined that many of these passwords are reasonable candidates for human-generated passwords. As such, we speculate that a possibly large number of passwords generated by PassGAN, that did not match our test sets, might still match user accounts from services other than RockYou and LinkedIn. We list a small sample of these passwords in Table 4.

5 Remarks

In this section, we summarize the findings from our experiments, and discuss their relevance in the context of password guessing.

Character-level GANs are well suited for generating password guesses. In our experiments, PassGAN was able to match 34.2% of the passwords in a testing set extracted from the RockYou password dataset, when trained on a different subset of RockYou. Further, we were able to match 21.9% of the password in the LinkedIn dataset when PassGAN was trained on the RockYou password set. This is remarkable because PassGAN was able to achieve these results with no additional information on the passwords that are present only in the testing dataset. In other words, PassGAN was able to correctly guess a large number of passwords that it did not observe given access to nothing more than a set of samples.

Current rule-based password guessing is very efficient but limited. In our experiments, rule-based systems were able to match or outperform other password guessing tools when the number of allowed guesses was small. This is a testament to the ability of skilled security experts to encode rules that generate correct matches with high probability. However, our experiments also confirmed that the main downside of rule-based password guessing is that rules can generate only a finite, relatively small set of passwords. In contrast, PassGAN was able to eventually surpass the number of matches achieved using password generation rules.

As a result, the best password guessing strategy is to use multiple tools. In our experiments, each password guessing approach has an edge in a different setting. Our results confirm that combining multiple techniques leads to the best overall performance. For instance, by combining the output of PassGAN with the output of the Best64 rules, we were able to match 48% of the passwords in the RockYou testing dataset (which represents a 50.8% increase in the number of matches) and 30.6% of the passwords from the LinkedIn dataset—an increase of about 73.3%. Given the current performance of both PassGAN and FLA, it is not unlikely that tools alone will soon be able to replace rule-based password guessing tools entirely.

GANs are expressive enough to generate passwords from Markovian processes, rules, and to capture more general password structures. Our experiments show that PassGAN is competitive with FLA, which treats password guessing primarily as a Markovian process. Without any knowledge of password rules or guidance on password structure, PassGAN was able to match the performance of FLA within an order of magnitude of guesses by leveraging only knowledge that it was able to extract from a limited number of samples. Further, because GANs are more general tools than Markov models, in our experiment PassGAN was able to generate matching passwords that were ranked as very unlikely by FLA, using a limited number of guesses.

GANs generalize well to password datasets other than their training dataset. When we evaluated PassGAN on a dataset (LinkedIn [35]) distinct from its training set (RockYou [57]), the drop in matching rate was modest, especially compared to other tools. Moreover, when tested on LinkedIn, PassGAN was able to match the other tools within a lower or equal number of guesses compared to RockYou.

State-of-the-art GANs density estimation is correct only for a subset of the space they generate. Our experiments show that IWGAN's density estimation matches the training set for high-frequency passwords. This is important because it allows PassGAN to generate highly-likely candidate passwords early. However, our experiments also show that as the frequency of a password decreases, the quality of PassGAN's density estimation deteriorates. While this becomes less relevant as PassGAN generates more passwords, it shows that the number of passwords that PassGAN needs to output to achieve a particular number of matches could significantly decrease if it is instantiated using a character-level GAN that performs more accurate density estimation. Similarly, a more extensive training dataset, coupled with a more complex neural network structure, could improve density estimation (and therefore PassGAN's performance) significantly.

Final Remarks. GANs estimate the density distribution of the training dataset. As a result, PassGAN outputs repeated password guesses. While a full brute-force guessing attack would have full coverage, learning from the training data distribution allows PassGAN to perform a more efficient attack by generating highly likely guesses. Because password generation can be performed offline, PassGAN could produce several billions of guesses beforehand, and store them

in a database. In our experiments, we stored unique password samples, and later used these samples for testing purposes, thus avoiding repetitions. If needed, Bloom filters with appropriate parameters could also be used to discard repeated entries, thus enabling efficient online password guessing.

Clearly, PassGAN can be used in a distributed setting, in which several instances independently output password guesses. While it is possible to avoid local repetitions using, e.g., Bloom filters, coordinating the removal of duplicates among different nodes is more complex and, potentially, more expensive. The appropriate way to address this problem depends primarily on three factors: (1) the cost of generating a password guess; (2) the cost of testing a password guess; and (3) the cost of synchronizing information about previously-generated password between nodes.

If the cost of generating passwords is less than the cost of testing them, and synchronization among nodes is not free, then avoiding repetitions across nodes is not essential. Therefore each model can sample without the need of being aware of other models' generated samples.

If the cost of testing password guesses is less than the cost of generating them, then it might be beneficial to periodically coordinate among nodes to determine which samples have been generated. The synchronization cost dictates the frequency of coordination.

Finally, PassGAN could significantly benefit and improve from new leaked password datasets. The model would improve by learning new rules, and the number of repeated samples could potentially be reduced.

6 Conclusion

In this paper, we introduced PassGAN, the first password guessing technique based on generative adversarial networks (GANs). PassGAN is designed to learn password distribution information from password leaks. As a result, unlike current password guessing tools, PassGAN does not rely on any additional information, such as explicit rules, or assumptions on the Markovian structure of user-chosen passwords. We believe that our approach to password guessing is revolutionary because PassGAN generates passwords with no user intervention—thus requiring no domain knowledge on passwords, nor manual analysis of password database leaks.

We evaluated PassGAN's performance by testing how well it can guess passwords that it was not trained on, and how the distribution of PassGAN's output approximates the distribution of real password leaks. Our results show that PassGAN is competitive with state-of-the-art password generation tools: in our experiments, PassGAN was always able to generate the same number of matches as the other password guessing tools.

However, PassGAN currently requires to output a larger number of passwords compared to other tools. We believe that this cost is negligible when considering the benefits of the proposed technique. Further, training PassGAN on a larger dataset enables the use of more complex neural network structures, and more

comprehensive training. As a result, the underlying GAN can perform more accurate density estimation, thus reducing the number of passwords needed to achieve a specific number of matches.

Changing the generative model behind PassGAN to a conditional GAN might improve password guessing in all scenarios in which the adversary knows a set of keywords commonly used by the user (e.g., the names of user's pets and family members). Given this knowledge, the adversary could condition the GAN to these particular words, thus enabling the generator to give special attention to a specific portion of the search space where these keywords reside.

PassGAN can potentially be used in the context of generating Honeywords [28]. Honeywords are decoy passwords that, when mixed with real passwords, substantially reduce the value of a password database for the adversary. Wang et al. [65], raised concerns about the previous techniques proposed by Juels et al. [28] to generate Honeywords: if Honeywords can be easily distinguished from real passwords, then their usefulness is significantly reduced. An extension of PassGAN could potentially address this problem and will be the subject of future work.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: a system for large-scale machine learning. In: OSDI. vol. 16, pp. 265–283 (2016)
2. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein GAN. CoRR **abs/1701.07875** (2017), `http://arxiv.org/abs/1701.07875`
3. Berthelot, D., Schumm, T., Metz, L.: BEGAN: Boundary equilibrium generative adversarial networks. arXiv preprint arXiv:1703.10717 (2017)
4. Binkowski, M., Sutherland, D., Arbel, M., Gretton, A.: Demystifying MMD GANs. International Conference on Learning Representations (ICLR) (2018)
5. Cao, Y., Ding, G.W., Lui, Y.C., Huang, R.: Improving GAN training via binarized representation entropy (BRE) regularization. International Conference on Learning Representations (ICLR) (2018)
6. Castelluccia, C., Dürmuth, M., Perito, D.: Adaptive password-strength meters from markov models. In: NDSS (2012)
7. Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., Abbeel, P.: Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In: Advances in Neural Information Processing Systems. pp. 2172–2180 (2016)
8. Ciaramella, A., D'Arco, P., De Santis, A., Galdi, C., Tagliaferri, R.: Neural network techniques for proactive password checking. IEEE Transactions on Dependable and Secure Computing **3**(4), 327–339 (2006)
9. Daskalakis, C., Ilyas, A., Syrgkanis, V., Zeng, H.: Training GANs with optimism. International Conference on Learning Representations (ICLR) (2018)
10. Dell'Amico, M., Michiardi, P., Roudier, Y.: Password strength: An empirical analysis. In: Proceedings IEEE INFOCOM. pp. 1–9. IEEE (2010)
11. Denton, E.L., Chintala, S., Fergus, R., et al.: Deep generative image models using a laplacian pyramid of adversarial networks. In: Advances in neural information processing systems. pp. 1486–1494 (2015)

12. Dorsey, B.: Markov-chain password generator. https://github.com/brannondorsey/markov-passwords (2017)
13. Duc, B., Fischer, S., Bigun, J.: Face authentication with gabor information on deformable graphs. IEEE Transactions on Image Processing **8**(4), 504–516 (1999)
14. Dürmuth, M., Angelstorf, F., Castelluccia, C., Perito, D., Abdelberi, C.: OMEN: Faster password guessing using an ordered markov enumerator. In: ESSoS. pp. 119–132. Springer (2015)
15. Fiegerman, S.: Yahoo says 500 million accounts stolen (2017), http://money.cnn.com/2016/09/22/technology/yahoo-data-breach/index.html
16. Frank, M., Biedert, R., Ma, E., Martinovic, I., Song, D.: Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. IEEE transactions on information forensics and security **8**(1), 136–148 (2013)
17. Golla, M.: Password guessing using recurrent neural networks - the missing manual. https://www.password-guessing.org/blog/post/cupslab-neural-network-cracking-manual/ (2017)
18. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)
19. Graves, A.: Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850 (2013)
20. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of wasserstein GANs. In: Advances in Neural Information Processing Systems. pp. 5767–5777 (2017)
21. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of wasserstein GANs - code. https://github.com/igul222/improved_wgan_training (2017)
22. HashCat: https://hashcat.net (2017)
23. HashCat: HashCat v5.0.0, advanced password recovery (2018), https://hashcat.net/forum/showthread.php?mode=linear&tid=7903&pid=42585
24. Hitaj, B., Gasti, P., Ateniese, G., Pérez-Cruz, F.: PassGAN: A deep learning approach for password guessing. In: NeurIPS 2018 Workshop on Security in Machine Learning. SECML'18, Montreal, CANADA (co-located with NeurIPS 2018) (2018)
25. Hjelm, R.D., Jacob, A.P., Trischler, A., Che, T., Cho, K., Bengio, Y.: Boundary seeking GANs. International Conference on Learning Representations (ICLR) (2018)
26. Hoang, Q., Nguyen, T.D., Le, T., Phung, D.: MGAN: Training generative adversarial nets with multiple generators. International Conference on Learning Representations (ICLR) (2018)
27. Hunt, T.: Here's why [insert thing here] is not a password killer. https://www.troyhunt.com/heres-why-insert-thing-here-is-not-a-password-killer/ (2018)
28. Juels, A., Rivest, R.L.: Honeywords: Making password-cracking detectable. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 145–160. ACM (2013)
29. Kim, T., Cha, M., Kim, H., Lee, J., Kim, J.: Learning to discover cross-domain relations with generative adversarial networks. arXiv preprint arXiv:1703.05192 (2017)
30. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

31. Lab, C.: Fast, lean, and accurate: Modeling password guessability using neural networks (source code). https://github.com/cupslab/neural_network_cracking (2016)
32. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural computation **1**(4), 541–551 (1989)
33. LeCun, Y., Boser, B.E., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W.E., Jackel, L.D.: Handwritten digit recognition with a back-propagation network. In: Advances in neural information processing systems. pp. 396–404 (1990)
34. Li, Y., Swersky, K., Zemel, R.: Generative moment matching networks. In: International Conference on Machine Learning. pp. 1718–1727 (2015)
35. LinkedIn: Linkedin, https://hashes.org/public.php
36. Ma, J., Yang, W., Luo, M., Li, N.: A study of probabilistic password models. In: IEEE Symposium on Security and Privacy (SP). pp. 689–704. IEEE (2014)
37. position Markov Chains, H.P.: https://www.trustwave.com/Resources/SpiderLabs-Blog/Hashcat-Per-Position-Markov-Chains/ (2017)
38. Melicher, W., Ur, B., Segreti, S.M., Komanduri, S., Bauer, L., Christin, N., Cranor, L.F.: Fast, lean, and accurate: Modeling password guessability using neural networks. In: USENIX Security Symposium. pp. 175–191 (2016)
39. Mirza, M., Osindero, S.: Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784 (2014)
40. Miyato, T., Koyama, M.: cGANs with projection discriminator. International Conference on Learning Representations (ICLR) (2018)
41. Morris, R., Thompson, K.: Password security: A case history. Communications of the ACM **22**(11), 594–597 (1979)
42. Mroueh, Y., Li, C.L., Sercu, T., Raj, A., Cheng, Y.: Sobolev GAN. International Conference on Learning Representations (ICLR) (2018)
43. Mroueh, Y., Sercu, T., Goel, V.: Mcgan: Mean and covariance feature matching GAN. In: Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. pp. 2527–2535 (2017), http://proceedings.mlr.press/v70/mroueh17a.html
44. Murphy, K.P.: Handbook of Information Security, Information Warfare, Social, Legal, and International Issues and Security Foundations. John Wiley & Sons (2006)
45. Murphy, K.P.: Machine Learning: A Probabilistic Perspective. MIT Press (2012)
46. Nagarajan, V., Kolter, J.Z.: Gradient descent GAN optimization is locally stable. In: Advances in Neural Information Processing Systems. pp. 5585–5595 (2017)
47. Narayanan, A., Shmatikov, V.: Fast dictionary attacks on passwords using time-space tradeoff. In: Proceedings of the 12th ACM conference on Computer and communications security. pp. 364–372. ACM (2005)
48. Nowozin, S., Cseke, B., Tomioka, R.: f-GAN: Training generative neural samplers using variational divergence minimization. In: Advances in Neural Information Processing Systems. pp. 271–279 (2016)
49. Percival, C., Josefsson, S.: The scrypt password-based key derivation function. Tech. rep. (2016)
50. Perez, S.: Google plans to bring password-free logins to android apps by year-end (2017), https://techcrunch.com/2016/05/23/google-plans-to-bring-password-free-logins-to-android-apps-by-year-end/
51. Petzka, H., Fischer, A., Lukovnikov, D.: On the regularization of wasserstein GANs. International Conference on Learning Representations (ICLR) (2018)

52. Project, T.P.: http://thepasswordproject.com/leaked_password_lists_and_dictionaries (2017)
53. Provos, N., Mazieres, D.: Bcrypt algorithm. In: USENIX (1999)
54. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. In: 4th International Conference on Learning Representations (2016)
55. the Ripper, J.: http://www.openwall.com/john/ (2017)
56. the Ripper Markov Generator, J.: http://openwall.info/wiki/john/markov (2017)
57. RockYou: Rockyou. http://downloads.skullsecurity.org/passwords/rockyou.txt.bz2 (2010)
58. Roth, K., Lucchi, A., Nowozin, S., Hofmann, T.: Stabilizing training of generative adversarial networks through regularization. In: Advances in Neural Information Processing Systems. pp. 2018–2028 (2017)
59. Rules, H.: https://github.com/hashcat/hashcat/tree/master/rules (2017)
60. Rules, J.T.R.K.: http://contest-2010.korelogic.com/rules.html (2017)
61. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. nature **323**(6088), 533 (1986)
62. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training gans. In: Advances in Neural Information Processing Systems. pp. 2234–2242 (2016)
63. Sitová, Z., Šeděnka, J., Yang, Q., Peng, G., Zhou, G., Gasti, P., Balagani, K.S.: HMOG: New behavioral biometric features for continuous authentication of smartphone users. IEEE Transactions on Information Forensics and Security **11**(5), 877–892 (2016)
64. Sutskever, I., Martens, J., Hinton, G.E.: Generating text with recurrent neural networks. In: Proceedings of the 28th International Conference on Machine Learning (ICML-11). pp. 1017–1024 (2011)
65. Wang, D., Cheng, H., Wang, P., Yan, J., Huang, X.: A security analysis of honeywords. NDSS (2018)
66. Wei, X., Gong, B., Liu, Z., Lu, W., Wang, L.: Improving the improved training of wasserstein GANs: A consistency term and its dual effect. International Conference on Learning Representations (ICLR) (2018)
67. Weir, M.: Probabilistic password cracker. https://sites.google.com/site/reusablesec/Home/password-cracking-tools/probablistic_cracker (2009)
68. Weir, M., Aggarwal, S., De Medeiros, B., Glodek, B.: Password cracking using probabilistic context-free grammars. In: 30th IEEE Symposium on Security and Privacy. pp. 391–405. IEEE (2009)
69. Wu, Y., Burda, Y., Salakhutdinov, R., Grosse, R.: On the quantitative analysis of decoder-based generative models. arXiv preprint arXiv:1611.04273 (2016)
70. Zhang, H., Xu, T., Li, H., Zhang, S., Huang, X., Wang, X., Metaxas, D.: Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. arXiv preprint arXiv:1612.03242 (2016)
71. Zhong, Y., Deng, Y., Jain, A.K.: Keystroke dynamics for user authentication. In: Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on. pp. 117–123. IEEE (2012)
72. Zhou, Z., Cai, H., Rong, S., Song, Y., Ren, K., Zhang, W., Wang, J., Yu, Y.: Activation maximization generative adversarial nets. International Conference on Learning Representations (ICLR) (2018)