PassGAN: A Deep Learning Approach for Password Guessing

Briland Hitaj*, Paolo Gasti†, Giuseppe Ateniese* and Fernando Perez-Cruz*
*Computer Science Department, Stevens Institute of Technology
Email: {bhitaj, gatenies, fperezcr}@stevens.edu
†Computer Science Department, New York Institute of Technology
Email: pgasti@nyit.edu

***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 straightforward dictionary attacks, these tools can expand dictionaries using password generation rules. These rules define transformations such as concatenation of words (e.g., "password123456") and *leet speak* (e.g., "password" becomes "p4s5w0rd"). Although these rules perform well on current password datasets, creating new rules that are optimized for new datasets is a laborious task that requires specialized expertise.**

In this paper, we devise how to replace human-generated password rules with a theory-grounded password generation approach based on machine learning. The result of this effort is PassGAN, a novel technique that leverages Generative Adversarial Networks (GANs) to enhance password guessing. PassGAN generates password guesses by training a GAN on a list of leaked passwords. Because the output of the GAN is distributed closely to its training set, the password generated using PassGAN are likely to match passwords that have not been leaked yet. PassGAN represents a substantial improvement on rule-based password generation tools because it infers password distribution information autonomously from password data rather than via manual analysis. As a result, it can effortlessly take advantage of new password leaks to generate richer password distributions.

Our experiments show that this approach is very promising. When we evaluated PassGAN on two large password datasets, we were able to outperform John the Ripper's SpyderLab rules by a 2x factor, on average, and we were competitive with the best64 and gen2 rules from HashCat—our results were within a 2x factor from HashCat's rules. More importantly, when we combined the output of PassGAN with the output of HashCat, we were able to match 18%-24% more passwords than HashCat alone. This is remarkable because it shows that PassGAN can generate a considerable number of passwords that are out of reach for current tools.

I. 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. Unfortunately, multiple password database leaks have shown that users tend to choose easy-to-guess passwords [14], [18], [42], primarily composed of common strings (e.g., `password`, `123456`, `iloveyou`), and variants thereof.

Password guessing tools provide a valuable tool for identifying weak passwords, especially when they are stored in hashed form [54], [58]. 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 [71] and HashCat [28], 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 constructed from a coherent and principled analysis of large password datasets. Further, developing and testing new heuristics is a time-consuming task that requires specialized expertise. To address these shortcomings, in this paper we propose PassGAN, a new approach for generating password guesses based on deep learning and Generative Adversarial Networks (GANs) [23]. GANs are recently-introduced machine-learning tools designed to perform density estimation in high-dimensional spaces [23]. A GAN is composed of two deep neural networks: a generative deep neural network (G), and a discriminative deep neural network (D). D is designed to distinguish between "real samples" from a distribution, and "fake samples", generated by G. The two deep neural networks interact with each other over many iterations. In each iteration, fake samples from G are given to D. The output of D is then provided to G, which uses it as feedback to generate fake samples that are distributed closer and closer to the real samples. After a sufficient number of iterations, the output of G simply becomes the output of the GAN. PassGAN leverages this technique to generate new password guesses. Our core idea is to train D using a list of leaked passwords (real samples). Therefore, at each iteration, the output of PassGAN (fake samples) becomes closer to the distribution of passwords in the original leak, and hence more likely to match real users' passwords. To the best of our knowledge, this work is the first to use GANs for this purpose.

PassGAN represents a principled and theory-grounded take on the generation of password guesses. We explore 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 GANs can generate high-quality password guesses. In our experiments, we were able to match 2,774,269 out of 5,919,936 passwords (46.86%) from a testing set composed of real user passwords from the RockYou dataset [61], and 4,996,980 out of 43,354,871 passwords (11.53%) from the LinkedIn dataset [40]. Further, the overwhelming majority of passwords generated by PassGAN that did not match our testing set still "looked like" human-generated passwords; (2) we show that our technique is competitive with state-of-the-art password generation rules. Even though these rules were specifically tuned for the dataset used in our evaluation, the quality of PassGAN's output was comparable to (in the case of HashCat), or better than (in the case of John the Ripper) that of password rules; (3) our results also show that PassGAN can be used to *complement* password generation rules. In our experiments, we successfully used PassGAN to generate password matches that were not generated by any password rule. When we combined the output of PassGAN with the output of HashCat, we were able to match between 18% and 24% additional unique passwords compared to HashCat alone; and (4) in contrast with password generation rules, PassGAN can generate a practically unbounded number of password guesses. Our experiments show that the number of new (unique) password guesses increases steadily with the overall number of passwords generated by the GAN. This is important because currently the number of unique passwords generated using rules is ultimately bounded by the size of the password dataset used to instantiate these rules.

We consider this work the first step towards a fully automated generation of high-quality password guesses. Our results constitute evidence that, when trained with large enough passwords datasets, and when instantiated with a complex enough neural network architecture, GANs can outperform rule-based password guessing. Moreover, PassGAN achieves this result while requiring none of the user effort commonly associated with the design of password guessing rules.

We argue that this work is relevant, important, and timely. *Relevant*, because despite numerous alternatives [55], [67], [21], [17], [80], 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 to make password-based systems more secure. And *timely*, because recent leaks containing hundreds of millions of passwords [20] provide a formidable source of data for attackers to compromise systems, and for system administrators to re-evaluate password policies.

Organization. The rest of this paper is organized as follows. In Section II, we briefly overview deep learning, GANs, and password guessing, and provide a summary of the relevant state of the art. Section III discusses the architecutral and training choices for the GAN used to instantiate PassGAN, and the hyperparameters used in our evaluation. We report on the evaluation of PassGAN, and on the comparison with state-of-the-art rule-based techniques, in Section IV. We conclude in Section V.

II. Background and Related Work

In this section, we present a brief overview of deep learning and GANs. We then review the state of the art in password guessing.

A. Deep Learning

In the mid-nineties, machine learning methods such as support vector machines [64], random forests [7], and Gaussian processes [60], showed remarkable results in classification and regression for mostly uncorrelated human engineered (hand-coded) features. Starting in the mid-2000s, with an increased availability of storage and data, these methods have been superseded by deep learning. Research on deep learning has shown that features can effectively be learned from data, and that hand-coded features tend to underperform learned features. These gains are more relevant with correlated features, in which human-engineered features might only encode low-dimensional correlations.

Deep learning is being extensively used in solving problems related to computer vision [39], image processing [70], video processing [16], [50], speech recognition [26], natural language processing [2], [12], [79] or gaming [24], [36], [45], [47]. Recently, there have also been significant improvements in using deep learning in health related problems [13], [19].

Deep learning has raised several privacy implications on data usage, on what can be learned from trained models, and on the ability of a model to learn more private information than necessary for the given task. For this reason, researchers have proposed privacy-preserving collaborative learning techniques [65], and techniques that rely on differential privacy [1]. However, recent work has shown that these techniques are not as privacy preserving as originally thought. Specifically, it was shown that trained models are susceptible to information leakage [4], model inversion attacks [22], membership attacks [66], [29], and model extraction attacks [74], even when the models are trained using privacy-preserving collaborative learning techniques [31].

In addition to attacks that extract information from trained models, it was recently shown that samples could be subtly modified so that they look unchanged to the human eye, but are consistently misclassified by deep learning algorithms [52], [51], [43], [8], [9], [35], [41], [30]. Several countermeasures have been proposed [53], [77]. However, this is still an open research problem.

B. Generative Adversarial Networks

Generative Adversarial Networks (GANs) represent a remarkable advance in the area of deep learning. A GAN is composed of two neural networks, a generative deep neural network G, and a discriminative deep neural network D. Given an input dataset $\mathcal{I} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, the goal of G is to produce "fake" samples from the underlying probability distribution $\Pr(\mathbf{x})$, that are accepted by D. At the same time,

D's goal is to learn to distinguish fake samples from G from the real ones coming from $\mathcal{I}$. More formally, on input a simple noise distribution $\mathbf{z}$, the optimization problem solved by GANs can be summarized as follows:

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

where the model attempts to minimize with respect to θ_G, and simultaneously maximize with respect to θ_D. The learning phase is considered complete when D is unable to distinguish between the fake samples produced by G, and the real samples from $\mathcal{I}$.

Since the original work by Goodfellow et al. [23], there have been several improvements on GANs. Radford et al. [59] introduce DCGAN, which improves on [23] by using a convolutional neural network instead of a multi-layer perceptron. As a result, DCGAN can produce more realistic image samples compared to [23].

Other work on GANs includes BEGAN [5], DiscoGAN [33], Conditional GAN [46], AdaGAN [73], InfoGAN [11], Laplacian Pyramid GAN [15], and StackGAN [78]. These techniques introduce improvements to prior work, such as new approaches to training and using the GAN.

Arjovsky et al. introduce the Wasserstein GAN (WGAN) [3]. WGAN improves learning stability of prior GANs by using gradient clipping. Benefits of this approach include reduced mode collapse, and meaningful learning curves, which are helpful in identifying optimal hyperparameters.

All work above focuses on the generation of realistic images. To address the problem of text generation, Gulrajani et al. [27] recently introduced the Improved Wasserstein GAN (IWGAN). With IWGAN, both G and D are simple convolutional neural networks (CNNs). G takes as input a latent noise vector, transforms it by forwarding it through its convolutional layers, and outputs a sequence of 32 one-hot character vectors. A *softmax* nonlinearity is applied at the output of G, and then forwarded to D. Each output character from IWGAN is obtained by computing the *argmax* of each output vector produced by G.

C. Password Guessing

In a password guessing attack, the adversary attempts to identify the password of one or more users by repeatedly testing multiple candidate passwords. Password guessing attacks are probably as old as password themselves [6], with more formal studies dating back to 1979 [48].

Two popular modern password guessing tools are John the Ripper (JTR) [71] and HashCat [28]. 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 [63], [62]; and Markov-model-based attacks [72], [56], in which each character of a password is selected via a stochastic process that considers one or more preceding character, and which is trained on dictionaries of plaintext passwords. JTR and HashCat are notably effective at guessing passwords. Specifically, there have been several instances in which well over 90% of the password leaked from online services have been successfully recovered [57].

Markov models were first used to generate password guesses by Narayanan et al. [49]. Their approach uses manually defined password rules, such as which portion of the generated passwords is composed of letters and numbers. This technique was subsequently improved by Weir et al. [75], who showed how to "learn" these rules from password distributions. This early work has been subsequently extended by Ma et al. [42] and by Durmuth et al. [18]. Techniques based on Markov models have also been used to implement real-time password strength estimators, and to evaluate the strength of passwords in plaintext databases (see, e.g., [14], [10]).

Probabilistic context-free grammars (PCFGs) [32], [75] leverage manually-encoded information on passwords structure to generate new guesses. This information can be implicit (e.g., a dictionary word followed by the user's birth date) or explicit (e.g., passwords are required to contain at least six characters, one capital letter, and one digit). Appropriate tokens are then randomly selected to build passwords from the resulting grammars.

Recently, Melicher et al. [44] introduced a password guessing method based on recurrent neural networks [25], [69]. With this technique, the neural network is trained using passwords leaked from several websites. During password generation, the neural network outputs one password character at a time. Each new character (including a special end-of-password character) is selected based on its probability given the current password, similarly to methods based on Markov models. (This technique was also used in [44] to perform real-time password strength estimation.) The evaluation presented in [44] shows that their technique outperforms PCFGs, Markov models, 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).

III. GAN Architecture and Hyperparameters

To leverage the ability of the GAN to effectively estimate the probability 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. [27]. The IWGAN implementation used in this paper relies on the ADAM optimizer [34] to minimize the training error, i.e., to reduce the mismatch between the output of the model and its training data.

Our model is characterized by the following hyper-parameters:

- **Batch size**, which represents the number of passwords from the training set that propagate through the GAN at each step of the optimizer.
- **Number of iterations**, which indicates how many times the GAN invokes its forward step and its back-propagation step [38], [37]. In each iteration, the GAN runs one generator iteration and one or more discriminator iterations.
- **Number of discriminator iterations per generator iteration**, which indicates how many iterations the generator performs in each GAN iteration.
- **Model dimensionality**, which represents the number of dimensions (weights) for each convolutional layer.
- **Gradient penalty coefficient** (λ), which specifies the penalty applied to the norm of the gradient of the discriminator with respect to its input [27]. Increasing this parameter leads to a more stable training of the GAN [27].
- **Output sequence length**, which indicates the maximum length of the strings generated by the generator (G henceforth).
- **Size of the input noise vector (seed)**, which determines how many random bits are fed as input to G for the purpose of generating samples.
- **Maximum number of examples**, which represents the maximum number of training items (passwords, in the case of PassGAN) to load.
- **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.

We instantiated our model with a *batch size* of 64. 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 (see analysis in Section IV-A). The *number of discriminator iterations per generative iteration* was set to 10, which is the default value used by IWGAN. We experimented using 5 residual layers for both the generator and the discriminator, with each of the layers in both deep neural network having 128 *dimensions*.

We set the *gradient penalty* to 10 and 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 (see Section IV-A). The *maximum number of examples* loaded by the GAN was set to the size of the entire training dataset. We set the *size of the noise vector* to 128 floating point numbers.

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. [27].

IV. Evaluation

In this section, we first present our training and testing procedure. We then report the results of our experiments and compare the output of PassGAN with that of password generation rules commonly used with JTR and HashCat.

Our experiments were run using the TensorFlow implementation of IWGAN. We used TensorFlow version 1.2.1 for GPUs, 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.

A. GAN 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, and HashCat on a large set of passwords from the RockYou leak [61]. Entries in this dataset represent a mixture of common and complex passwords because these passwords were stored on servers in plaintext, and therefore all of them were recovered. We then counted how many of the passwords that each tool generated were present in two separate testing sets: a subset of RockYou distinct from the training set, and the LinkedIn password dataset [40].

The RockYou dataset contains 32,603,388 passwords. We selected all passwords of length 10 characters or less (29,599,680 passwords, which corresponds to 90.8% of the dataset), and used 80% of them (23,679,744 total passwords, 9,925,896 unique passwords) to train each tool. For testing, we used the remaining 20% (5,919,936 total passwords, 3,094,199 unique passwords).

We also tested each tool on all passwords of length 10 characters or less from the LinkedIn dataset. This dataset consists of 60,065,486 total unique passwords, out of which 43,354,871 unique passwords are of length 10 characters or less (frequency count was 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. Our results, presented in Section IV-B, suggest that the rules and the datasets used to recover the LinkedIn passwords substantially overlap with the rules and the dataset used in this work.

Our training and testing procedures allowed us to determine: (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.

Effects of the GAN Training Process on Its Output. 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. However, increasing the number of steps also increases the probability of overfitting [23], [76]. To evaluate this tradeoff on password data, we stored intermediate training checkpoints and generated 10^8 passwords at each checkpoint.

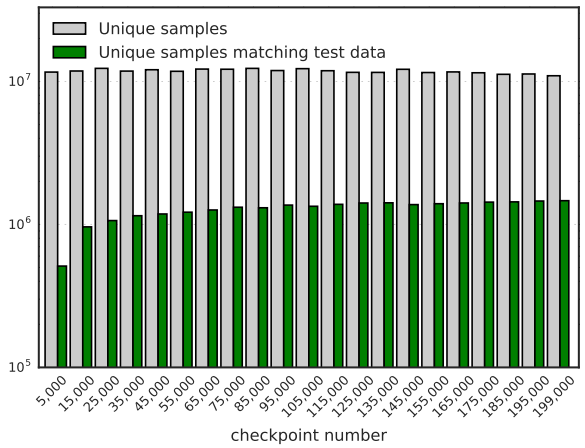

Fig. 1: Number of unique passwords generated by the GAN, and number of passwords matching the RockYou testing set. The x axis represents the number of iterations (checkpoints) of PassGAN's training process. For each checkpoint, PassGAN generated a total of 10^8 passwords.

Figure 1 shows the number of unique passwords generated by the GAN at each checkpoint, and how many of these passwords match with the content of the RockYou testing set. The number of unique samples generated by the GAN remains fairly constant with the number of iterations (checkpoint number). However, the number of passwords that match the testing set steadily increases with the number of iterations. This increase tapers off around 175,000-199,000 iterations, where the number of unique passwords also decreases slightly. 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 our RockYou training set.

B. Evaluating the Passwords Generated by PassGAN

We generated up to 10^{10} passwords using PassGAN, JTR, and HashCat. For JTR we used the SpiderLabs mangling rules [68], while for HashCat we used the best64 and gen2 rules [28]. These rules are commonly used in the password guessing literature [44], and have been optimized over the years on password datasets including RockYou and LinkedIn. Because of these dataset-specific optimizations, we consider these rules a good representation of the best matching performance that can be obtained using manually-generated rules.

Both best64 and gen2 were able to generate less than 10^{10} passwords (roughly 998M and 754M passwords, respectively—see Table I), given our training dataset. For the SpiderLabs mangling rules, we generated about $6 \cdot 10^{10}$ passwords. From this set, we sampled uniformly 528,834,530 unique passwords. This allowed us to perform a fair comparison between JTR and PassGAN because the latter generated

TABLE I: Comparison of uniqueness and novelty of the passwords generated using PassGAN, HashCat, and JTR. Column (1) shows the total number of passwords generated using each tool. In our experiments, HashCat was unable to generate as many passwords as PassGAN and JTR given the training dataset and the rule sets we used. Column (2) shows how many of the passwords generated by each tool were unique. Column (3) indicates how many unique passwords generated by each tool were already present in the training set.

Password generation tool	(1) Total passwords generated	(2) Unique passwords	(3) Passwords matched in the training set (9,926,278 entries)
PassGAN	10^6	182,036	27,320 (0.28%)
	10^7	1,357,874	134,647 (1.36%)
	10^8	10,969,748	487,878 (4.92%)
	10^9	80,245,649	1,188,152 (12%)
	$\approx 10^{9.86}$	441,357,719	1,825,915 (18.4%)
	10^{10}	528,834,530	2,177,423 (21.9%)
Hashcat best64 rules	754,315,842	441,357,719	9,898,464 (99.7%)
Hashcat gen2 rules	998,076,164	646,401,854	3,267,236 (32.9%)
JTR SpyderLabs rules	$6 \cdot 10^{10}$	528,834,530	2,278,045 (23%)
best64+GAN	**10,754,315,842**	**947,606,924**	**9,898,807 (99.7%)**

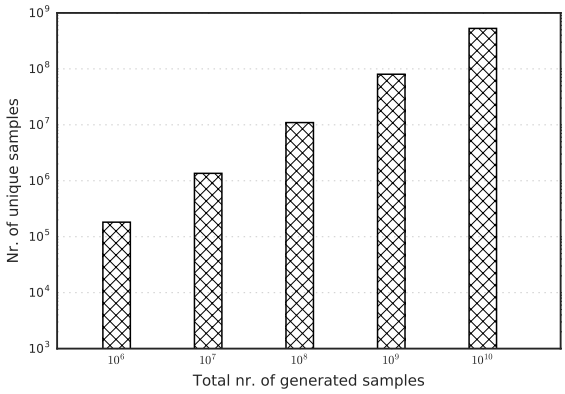

Fig. 2: Number of unique samples, compared to the total number of samples generated by PassGAN.

the same number of unique passwords from a set of 10^{10} samples.

We first determined whether increasing the number of passwords generated by PassGAN also increases the number of unique passwords, as well as the number of matches with the training and the testing sets. To this end, we generated various passwords sets of sizes between 10^6 and 10^{10}. We observed that, as the number of passwords increased, so did the number of unique passwords. Results of this evaluation are reported in Table I, column (2). As shown in Figure 2, 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 Figure 3, diminished as the number

TABLE II: Comparison of passwords generated using our technique, HashCat, and JTR, on the RockYou testing set. Column (2) shows how many of generated passwords appear in the testing set, which is composed of 3,094,199 unique entries. Column (3) shows the number of generated passwords that appear in the testing set, weighting each match by the number of corresponding entries in the testing set. (The testing set contains 5,919,936 non-unique entries). Columns (4) and (5) report the number of generated passwords that appear in the testing set, and that were not in the training set (i.e., *new passwords*). The numbers in Column (4) count each match as unique, while the numbers in Column (5) weight each password match by the number of occurrence of the password in the test set. The maximum number of matches is 1,978,367 in Column (4) and 2,032,728 in Column (5).

Password generation tool	(1) Unique passwords generated	(2) Passwords matched in testing set (unique)	(3) Passwords matched in testing set (with repetitions)	(4) Passwords matched in testing, and not in training set (unique)	(5) Passwords matched in testing set, and not in training set (with repetitions)
PassGAN	182,036	17,421 (0.56%)	449,583 (7.59%)	1,850 (0.094%)	2,039 (0.1%)
	1,357,874	76,473 (2.47%)	870,126 (14.7%)	11,398 (0.576%)	12,489 (0.6%)
	10,969,748	236,375 (7.64%)	1,466,336 (24.8%)	54,325 (2.746%)	58,682 (2.88%)
	80,245,649	501,272 (16.2%)	2,133,147 (36%)	162,652 (8.221%)	172,997 (8.51%)
	441,357,719	699,798 (22.6%)	2,373,825 (40.1%)	286,736 (14.49%)	301,416 (14.83%)
	528,834,530	833,434 (26.9%)	2,774,269 (46.9%)	342,439 (17.31%)	359,980 (17.7%)
Hashcat best64 rules	441,357,719	1,744,127 (56.4%)	4,545,600 (76.8%)	630,067 (31.85%)	662,215 (32.577%)
Hashcat gen2 rules	646,401,854	1,288,769 (41.7%)	4,060,366 (68.6%)	448,969 (22.69%)	475,462 (23.39%)
JTR SpyderLabs rules	528,834,530	472,417 (15.3%)	1,368,106 (23.1%)	161,807 (8.178%)	170,437 (8.38%)
best64+GAN	**947,606,924**	**1,859,765 (60.1%)**	**4,664,141 (78.8%)**	**745,680 (37.69%)**	**780,705 (38.4%)**

TABLE III: Comparison of passwords generated using our technique, HashCat, and JTR, on the LinkedIn testing set, which contains 43,354,871 unique entries. Column (2) shows how many of the generated passwords appear in the testing set, while Column (3) shows how many generated passwords match the testing set, and were not in the training set (the maximum number of matches is 40,597,129).

Password generation tool	(1) Unique passwords generated	(2) Passwords matched in testing set (unique)	(3) Passwords matched in testing set, and not in training set (unique)
PassGAN	182,036	33,794 (0.08%)	12,946 (0.032%)
	1,357,874	185,775 (0.43%)	87,230 (0.215%)
	10,969,748	804,326 (1.86%)	474,861 (1.169%)
	80,245,649	2,341,529 (5.40%)	1,637,122 (4.032%)
	441,357,719	4,185,625 (9.65%)	3,255,417 (8.018%)
	528,834,530	4,996,980 (11.5%)	3,890,043 (9.582%)
Hashcat best64 rules	441,357,719	9,930,005 (22.9%)	7,174,986 (17.67%)
Hashcat gen2 rules	646,401,854	6,271,492 (14.5%)	4,475,775 (11.02%)
JTR SpyderLabs rules	528,834,530	2,763,640 (6.37%)	2,023,113 (4.98%)
best64+GAN	**947,606,924**	**11,702,590 (27%)**	**8,947,510 (22.039%)**

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.

Our experiments also show that PassGAN's output has a higher rate of repeated passwords compared to JTR and HashCat rules. This is since, of the three tools, PassGAN is the only tool that attempts to generate passwords with the same distribution as the training set—which is characterized by a large number of repeated passwords. This allows PassGAN to output highly likely passwords earlier than less likely

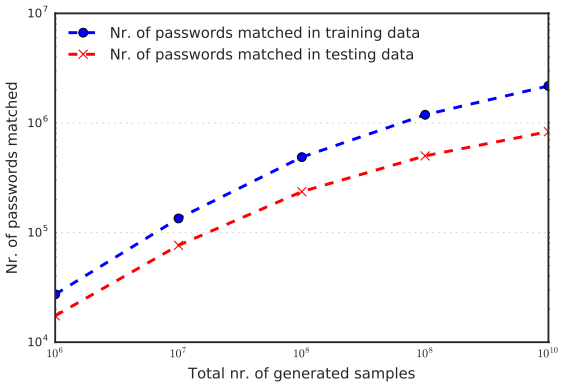

Fig. 3: Number of passwords generated by the GAN that appear in the training and in the testing set, as the number samples in the GAN's output increases.

passwords, thus potentially reducing the number of guessing attempts in practice.

For each tool, we report the number of passwords generated that also appear in the training set. We do not consider these passwords to be valuable because any match they lead to could have been trivially obtained via a dictionary attack based on the training set. Table I, column (3), shows the results for each tool, and Figure 3 (blue line) provides further details related to PassGAN's output. In our experiments, PassGAN generated considerably fewer passwords that matched its training set, compared to both JTR and HashCat.

Next, we calculated the number of passwords, generated by each tool, which matched the RockYou and LinkedIn testing sets. Our results for RockYou are presented in Table II, while

TABLE IV: Sample of passwords generated by the GAN that did not match the testing set.

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

results for LinkedIn are shown in Table III. Columns (2) and (3) of Table II show the number of passwords in the RockYou testing set that were generated by each tool—without counting repeated matches *and* counting repeated matches, respectively. Columns (4) and (5) of Table II report the number of matches when we excluded passwords that appear in both the training set and the testing set (without and with repeated matches, respectively). In our experiments, the best64 rules were able to match 54% more passwords than the GAN, with best64 outperforming our technique by roughly a 1.7×-2× factor. In the same experiments, PassGAN significantly and consistently outperformed the SpiderLabs rules used with JTR.

Under all metrics, combining the output of the GAN with the output of best64 led to the best performance, resulting in 115,613 additional passwords being matched in the RockYou testing set when counting unique passwords (see Table II, Column (4)), and 118,490 when counting repetitions (See Table II, Column (5)). In both cases, this corresponds to an 18% improvement over HashCat's best64. We consider this is a significant result because it shows that the GAN was able to successfully model how users choose their passwords based only on password samples—in some case better than the experts that wrote HashCat's password generation rules.

Columns (2) and (3) of Table III show the number of matches with the LinkedIn testing set. Column (2) includes all matches, while Column (3) does not include matches for passwords that also appear in the RockYou training set. In our experiments, the gap between HashCat's rules and PassGAN was smaller than with the RockYou dataset. This shows that the GAN generalizes better than manually-generated rules. Further, combining PassGAN with best64 led once again to the highest number of matches. These results show that PassGAN provides substantial benefits when no samples from the target distribution are available, which is often the case with password guessing. Further, when we combined the output of PassGAN with that of HashCat's best64 rules, we were able to match 1,772,524 more passwords than with best64 alone—an increase of about 24%.

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 the GAN, and that did not match our test sets, might still match user accounts from services other than RockYou. We list a small sample of these passwords in Table IV.

V. Conclusion

In this paper, we introduced PassGAN, the first password guessing technique based on GANs. Our results show that character-level GANs are well suited for generating password guesses when trained on leaked password data. Further, our results show that GANs generalize well when trained on a password dataset, and tested on a differently distributed dataset.

In our experiments, we were able to match over 46% of the passwords in a testing set extracted from the RockYou password dataset, when the GAN was trained on a different subset of RockYou. Compared to rule-based password generation techniques, our experiments show that PassGAN is very competitive. Although HashCat's best64 and gen2 rules outperformed PassGAN, our approach was able to match twice as many passwords as JTR's SpyderLabs rules. Moreover, by combining the output of the GAN with the output of the best64 rules, we were able to match more than 78% of the passwords in the RockYou dataset, and almost 27% of the passwords from the LinkedIn dataset—an increase of about 18% and 24%, respectively. This is remarkable because it shows that the GAN can generate a large number of passwords that are out of reach for current state-of-the-art password generation tools. Further, when we evaluated each password guessing tool on a dataset (LinkedIn) distinct from the training set (RockYou), the drop in matching rate of the GAN was less pronounced.

We believe that our approach to password guessing is revolutionary because, unlike current rule-based tools, PassGAN was able to generate passwords with no user intervention—thus requiring no domain knowledge on passwords, nor manual analysis of password database leaks. Also, our evaluation of training performance suggests that, when supplied with a large enough leaked password set, the performance of PassGAN could surpass that of the best rule-based password generation techniques.

References

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 308–318.

[2] A. Abdulkader, A. Lakshmiratan, and J. Zhang. (2016) Introducing deeptext: Facebook's text understanding engine. [Online]. Available: https://tinyurl.com/jj359dv

[3] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *CoRR*, vol. abs/1701.07875, 2017.

[4] G. Ateniese, L. V. Mancini, A. Spognardi, A. Villani, D. Vitali, and G. Felici, "Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers," *International Journal of Security and Networks*, vol. 10, no. 3, pp. 137–150, 2015.

[5] D. Berthelot, T. Schumm, and L. Metz, "Began: Boundary equilibrium generative adversarial networks," *arXiv preprint arXiv:1703.10717*, 2017.

[6] H. Bidgoli, "Handbook of information security threats, vulnerabilities, prevention, detection, and management volume 3," 2006.

[7] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[8] N. Carlini and D. Wagner, "Defensive distillation is not robust to adversarial examples," *arXiv preprint arXiv:1607.04311*, 2016.

[9] ——, "Adversarial examples are not easily detected: Bypassing ten detection methods," *arXiv preprint arXiv:1705.07263*, 2017.

[10] C. Castelluccia, M. Dürmuth, and D. Perito, "Adaptive password-strength meters from markov models." in *NDSS*, 2012.

[11] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2016, pp. 2172–2180.

[12] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[13] A. A. Cruz-Roa, J. E. A. Ovalle, A. Madabhushi, and F. A. G. Osorio, "A deep learning architecture for image representation, visual interpretability and automated basal-cell carcinoma cancer detection," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer Berlin Heidelberg, 2013, pp. 403–410.

[14] M. Dell'Amico, P. Michiardi, and Y. Roudier, "Password strength: An empirical analysis," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[15] E. L. Denton, S. Chintala, R. Fergus *et al.*, "Deep generative image models using a? laplacian pyramid of adversarial networks," in *Advances in neural information processing systems*, 2015, pp. 1486–1494.

[16] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2625–2634, 2015.

[17] B. Duc, S. Fischer, and J. Bigun, "Face authentication with gabor information on deformable graphs," *IEEE Transactions on Image Processing*, vol. 8, no. 4, pp. 504–516, 1999.

[18] M. Dürmuth, F. Angelstorf, C. Castelluccia, D. Perito, and C. Abdelberi, "Omen: Faster password guessing using an ordered markov enumerator." in *ESSoS*. Springer, 2015, pp. 119–132.

[19] R. Fakoor, F. Ladhak, A. Nazi, and M. Huber, "Using deep learning to enhance cancer diagnosis and classification," in *The 30th International Conference on Machine Learning (ICML 2013),WHEALTH workshop*, 2013.

[20] S. Fiegerman. (2017) Yahoo says 500 million accounts stolen. [Online]. Available: http://money.cnn.com/2016/09/22/technology/yahoo-data-breach/index.html

[21] M. Frank, R. Biedert, E. Ma, I. Martinovic, and D. Song, "Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication," *IEEE transactions on information forensics and security*, vol. 8, no. 1, pp. 136–148, 2013.

[22] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1322–1333.

[23] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[24] Google DeepMind. (2016) Alphago, the first computer program to ever beat a professional player at the game of GO. [Online]. Available: https://deepmind.com/alpha-go

[25] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[26] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[27] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," *CoRR*, vol. abs/1704.00028, 2017.

[28] HashCat. (2017). [Online]. Available: https://hashcat.net

[29] J. Hayes, L. Melis, G. Danezis, and E. D. Cristofaro, "LOGAN: Evaluating privacy leakage of generative models using generative adversarial networks," *CoRR*, vol. abs/1705.07663, 2017.

[30] W. He, J. Wei, X. Chen, N. Carlini, and D. Song, "Adversarial example defenses: Ensembles of weak defenses are not strong," *arXiv preprint arXiv:1706.04701*, 2017.

[31] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the GAN: Information leakage from collaborative deep learning," *CCS'17*, 2017.

[32] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, and J. Lopez, "Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms," in *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012, pp. 523–537.

[33] T. Kim, M. Cha, H. Kim, J. Lee, and J. Kim, "Learning to discover cross-domain relations with generative adversarial networks," *arXiv preprint arXiv:1703.05192*, 2017.

[34] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[35] J. Kos and D. Song, "Delving into adversarial attacks on deep policies," *arXiv preprint arXiv:1705.06452*, 2017.

[36] M. Lai, "Giraffe: Using deep reinforcement learning to play chess," *arXiv preprint arXiv:1509.01549*, 2015.

[37] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in neural information processing systems 2, NIPS 1989*. Morgan Kaufmann Publishers, 1990, pp. 396–404.

[38] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[39] Y. LeCun, K. Kavukcuoglu, C. Farabet *et al.*, "Convolutional networks and applications in vision." in *ISCAS*, 2010, pp. 253–256.

[40] LinkedIn. Linkedin. [Online]. Available: https://hashes.org/public.php

[41] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," *arXiv preprint arXiv:1611.02770*, 2016.

[42] J. Ma, W. Yang, M. Luo, and N. Li, "A study of probabilistic password models," in *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 2014, pp. 689–704.

[43] P. McDaniel, N. Papernot, and Z. B. Celik, "Machine learning in adversarial settings," *IEEE Security & Privacy*, vol. 14, no. 3, pp. 68–72, 2016.

[44] W. Melicher, B. Ur, S. M. Segreti, S. Komanduri, L. Bauer, N. Christin, and L. F. Cranor, "Fast, lean, and accurate: Modeling password guessability using neural networks," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 175–191. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/melicher

[45] C. Metz. (2016) Google's GO victory is just a glimpse of how powerful ai will be. [Online]. Available: https://tinyurl.com/l6ddhg9

[46] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[47] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[48] R. Morris and K. Thompson, "Password security: A case history," *Communications of the ACM*, vol. 22, no. 11, pp. 594–597, 1979.

[49] A. Narayanan and V. Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff," in *Proceedings of the 12th ACM conference on Computer and communications security*. ACM, 2005, pp. 364–372.

[50] Y. Pan, T. Mei, T. Yao, H. Li, and Y. Rui, "Jointly modeling embedding and translation to bridge video and language," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4594–4602, 2016.

[51] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," *arXiv preprint arXiv:1605.07277*, 2016.

[52] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proceedings of the 1st IEEE European Symposium on Security and Privacy*, 2015.

[53] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proceedings of the 37th IEEE Symposium on Security and Privacy*, 2015.

[54] C. Percival and S. Josefsson, "The scrypt password-based key derivation function," Tech. Rep., 2016.

[55] S. Perez. (2017) Google plans to bring password-free logins to android apps by year-end. [Online]. Available: https://techcrunch.com/2016/05/23/google-plans-to-bring-password-free-logins-to-android-apps-by-year-end/

[56] H. P. position Markov Chains. (2017). [Online]. Available: https://www.trustwave.com/Resources/SpiderLabs-Blog/Hashcat-Per-Position-Markov-Chains/

[57] T. P. Project. (2017). [Online]. Available: http://thepasswordproject.com/leaked_password_lists_and_dictionaries

[58] N. Provos and D. Mazieres, "Bcrypt algorithm," in *USENIX*, 1999.

[59] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *4th International Conference on Learning Representations*, 2016.

[60] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*. MIT press Cambridge, 2006, vol. 1.

[61] RockYou. (2010) Rockyou. [Online]. Available: http://downloads.skullsecurity.org/passwords/rockyou.txt.bz2

[62] H. Rules. (2017). [Online]. Available: https://github.com/hashcat/hashcat/tree/master/rules

[63] J. T. R. K. Rules. (2017). [Online]. Available: http://contest-2010.korelogic.com/rules.html

[64] B. Schölkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[65] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM, 2015, pp. 1310–1321.

[66] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 3–18.

[67] Z. Sitová, J. Šeděnka, Q. Yang, G. Peng, G. Zhou, P. Gasti, and K. S. Balagani, "Hmog: New behavioral biometric features for continuous authentication of smartphone users," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 5, pp. 877–892, 2016.

[68] T. SPIDERLABS. (2012) Korelogic-rules. [Online]. Available: https://github.com/SpiderLabs/KoreLogic-Rules

[69] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 1017–1024.

[70] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 1701–1708. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2014.220

[71] J. the Ripper. (2017). [Online]. Available: http://www.openwall.com/john/

[72] J. the Ripper Markov Generator. (2017). [Online]. Available: http://openwall.info/wiki/john/markov

[73] I. Tolstikhin, S. Gelly, O. Bousquet, C.-J. Simon-Gabriel, and B. Schölkopf, "Adagan: Boosting generative models," *arXiv preprint arXiv:1701.02386*, 2017.

[74] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis." in *USENIX*, 2016.

[75] M. Weir, S. Aggarwal, B. De Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *Security and Privacy, 2009 30th IEEE Symposium on*. IEEE, 2009, pp. 391–405.

[76] Y. Wu, Y. Burda, R. Salakhutdinov, and R. Grosse, "On the quantitative analysis of decoder-based generative models," *arXiv preprint arXiv:1611.04273*, 2016.

[77] V. Zantedeschi, M.-I. Nicolae, and A. Rawat, "Efficient defenses against adversarial attacks," *arXiv preprint arXiv:1707.06728*, 2017.

[78] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas, "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks," *arXiv preprint arXiv:1612.03242*, 2016.

[79] X. Zhang and Y. A. LeCun, "Text understanding from scratch," *arXiv preprint arXiv:1502.01710v5*, 2016.

[80] Y. Zhong, Y. Deng, and A. K. Jain, "Keystroke dynamics for user authentication," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*. IEEE, 2012, pp. 117–123.