Journal of Network and Systems Management (2022) 30:13
https://doi.org/10.1007/s10922-021-09620-w

PassMon: A Technique for Password Generation and Strength Estimation

Sanjay Murmu[1] · Harsh Kasyap[1] · Somanath Tripathy[1]

Received: 17 December 2020 / Revised: 8 June 2021 / Accepted: 9 August 2021 /
Published online: 17 October 2021

Abstract

The password is the most prevalent and reliant mode of authentication by date. We often come across many websites with user registration pages having different password strength estimation techniques. Most of them run lightweight java-script-based rules on the client-side, while others take it to the server and evaluate. The same password is measured on different scales and is treated as invalid, weak, medium, or strong by different meters. These constraints compel users to choose weak passwords. The state-of-the-art password guessing and strength estimating techniques are trained on the publicly available leaked data sets. They are able to cope with the dictionary attacks but became prone to adversarial attacks. Creating dynamic rules for such attacks is tedious and infeasible. This paper proposes an ensemble approach with a classification and guessing strategy. We devise a bi-directional generative adversarial network based algorithm to generate personalized passwords with an improved convergence rate. It generates as many numbers of samples compared to GAN in less time. The one-class SVM is trained over the leaked and generated passwords to estimate password strength. The passwords mainly comprise the medium and weak category, and it gives better performance drawing a similarity between weak passwords. LSTM has been tuned to predict the difficulty level to crack the given test password. Based on their combined results, the password strength is determined. This paper also proposes three password design methods to create memorable and reasonably strong passwords. They are simple to design by taking user personal information and adding randomness based on functional patterns.

Keywords Password policy · Password security · Adversarial password generation · BiGAN · Password strength

✉ Harsh Kasyap
harsh_1921cs01@iitp.ac.in

Extended author information available on the last page of the article

1 Introduction

Passwords have been the primary source of authentication since we started securing our resources. Technology leapfrogged to new advances, with everyone surrounded by devices with various applications running over the internet. They all store sensitive information and, therefore, require security services for authentic usage. Meanwhile, recent advancements in IoT have propelled various other mechanisms like pins [1], fingerprints [2], and face locks [3]. They face severe issues of hardware and software constraints. Among all these user-friendly techniques, the text-based password remains the most prevalent method as it is easy to implement, and no special hardware is required.

With cheap computation and readily available resources, passwords have become vulnerable and susceptible to various attacks. User often choose or concatenate common texts (e.g., “password”, “123456”) and their leet transformation. Usually, they tend to choose a password based on their personally identifiable information (PII) like name, dob, age, phone number, or organization. The user-chosen passwords are vulnerable to guessing attacks is because passwords follow Zipf’s law, a very skewed distribution [4]. However, Zipf’s law also mentions that the guess’s success rate decreases as this guess increases in the guess list. Social exposure of individuals and organizational information together makes it easy to guess one’s password. Attackers learn over the multiple leaked data sets and devise a smart password cracking strategy. Wang et al. [5] conducted an empirical study of many password creation policies over high-profile web services and found them vulnerable to targeted online guessing attacks. Another issue arises from the non-uniformity of password strength estimation and their approach across different applications. Many of the applications adopt a rule-based evaluation strategy, that can be easily analyzed by a chrome debugger. Some make it tough by processing the password at their server and pushing back the result. Existing password strength, guessing, and generation techniques suffer from easy prediction, frequent attacks, and massive computation. The prevailing estimation techniques do not account for smart rules to omit sensitive information and simple structures. Hence, they fail to mitigate targeted and personalized attacks.

The above study motivates us to design an effective approach for generating passwords and measuring their strength on a uniform scale. The art of evaluating password strength started using basic rules of using chars in different cases, digits, and special chars. New estimation techniques [6] were designed using some mathematical inference and theoretical computer science using Probabilistic grammars and Markov models [7].

Brute force and dictionary attacks were the basic attacks to guess the passwords and had a fair success rate. Researchers enhanced guessing attacks using probabilistic context-free grammar (PCFG) [8]. It calculates the probability of the frequency of the structure and guesses a password. For example, “password123” may be more probable than “P@ssW0rd!23”, depending on the password creation policy. The dynamics of research changed after the evolution of machines and deep learning. Many works have been proposed applying different classification and perceptron techniques. Machine learning techniques heavily rely on smart feature processing and extraction mechanism to learn from the data available. Deep learning does both feature extraction and modeling on a massive amount of data. All these techniques work well with existing databases but suffer from noisy and adversarially created samples.

Limitations of trained models on human-generated passwords have laid the requirement for generating new personalized passwords. Training with these samples would help to mitigate unseen attacks and evaluate the passwords on a substantial scale. John the Ripper[1] and HashCat[2] are the state-of-the-art techniques to generate new, highly likely passwords. Generative Adversarial Network (GAN) has been widely used to generate new and unseen samples [9]. In GAN, we find the distribution of the data set $P(w)$. With the help of a generator and discriminator, we try to create a distribution with noise(o), similar to $P(w)$. The existing methodology takes a longer time to converge the distribution. PASSGAN [10] is a GAN based theory-grounded password generation approach. It takes a large no of epochs to converge and a long time to generate passwords. Bi-directional GAN (BiGAN) is an extension of GAN and performs faster in comparison [11]. It converges much faster by adding one more component and gives satisfactory results for generating new passwords.

Different service providers, social sites, and corporates also suffer from framing rules for a secure and memorable password. The keyspace of the password and its corresponding entropy has been widely considered to design strong policies. However, Weir et al. [12] performed an extensive study and proved entropy as ineffective. Poor combinations of large key spaces suffer from offline dictionary attacks. We have tried to address these issues by putting the different spaces based on a random function to generate a strong and memorable password. Wang et al. [13, 14] have discussed the vulnerabilities arising due to the usage of PII (name, birthday) and sister passwords across different websites. Authors in [15] have discussed the difficulties of implementation by considering the above issues.

This work proposes a **PassMon: Password Monitor** incorporating password estimation, cracking, and generation strategies. We predict passwords' complexity by guessing and evaluating their strength on a uniform scale and likely generated password data sets. The major contributions are as follows.

- This paper devised an effective deep learning approach using bi-directional GAN for generating new highly likely passwords. It has a faster convergence rate compared to GAN.
- One-class SVM has been trained to evaluate password strength. It draws test password similarity with most of the weak passwords in the dataset.
- LSTM has been tuned for guessing the difficulty to crack the password by calculating the low probability ranking of prediction.

[1] https://www.openwall.com/john/.
[2] https://hashcat.net/hashcat/.

– New password design methods and compliances have been proposed and compared. They are easy to remember and based on user-related inputs by adding a fair amount of randomness on functional patterns.

The remainder of this paper is organized as follows. Section 2 discusses the related works. Section 3 briefs the preliminaries of bi-directional GAN and machine learning models used in this experiment. Section 4 discusses password design methods to create a strong and memorable password. Section 5 discusses the security model and the capabilities of the adversary. Section 6 summarizes the proposed framework and methodology. Section 7 lists the experiment setup, configuration and architecture. Section 8 illustrates the result and comparison with existing strength meters. Section 9 concludes and briefs the scope for future work.

2 Related Work

The science of password strength estimation evolved with the increased capabilities of attackers. Several techniques based on mathematical, theoretical computer science to machine learning have been proposed to evaluate the strength of a given password [16–18]. Researchers suggested different guessing strategies to crack the password.

Brute-Force and Dictionary Attack are the most traditional password guessing tools [19] where the attacker obtains the hash of the victim's password. It then tries to retrieve the plain text by guessing for thousands and a million times. It takes an ample amount of time and storage [20]. In a dictionary attack, the attacker tries to guess the password by adding new words to the input by applying pre-selected mangling rules. It requires the original word to be in the attacker's input dictionary to use the correct word-mangling rules [21]. The main challenge is to generate an ordered list of password guesses that match the victim's password. Authors in [22] created a strong password by salting durations of keystrokes and latencies between them.

The user often chooses weak passwords tuning the password policy using a common dictionary word adding some special character or numbers. This password can easily be compromised if the attacker has the user details. With this fact, researchers created three different password policies for generating secure and easy-to-remember passwords with diverse groups of experiment [23]. They proposed three strategies and evaluated them on various measures as Pearson's correlation and password compliance. Method one takes a random text and number. It changes the characters into uppercase based on the number's digits and appends them in the password. Method 2 proposes a random combination of letters and keyboard characters, which mixes the string with a random number taken as input from the user. Method three suggests choosing a favorite phrase or quote and make the alternate characters uppercase. Woods et al. [24] performed their study on the number of verification. They proved that verifying passwords three times can increase password memorability from 42% while 17% improvement two times. Though, it does not have any effect on memorability.

Markov model is based on the Markov chain rule, which is a state-based mathematical representation. It maps the transition of one state to another state according to specific probabilistic rules. For password generation, it uses the idea of a transition between the characters or group of characters [16]. If we take the word ‘Alice’ and use the Markov model of 3rd order, it gets divided into ‘Ali’, ‘lic’. The idea of a transition between the characters itself becomes its limitation as we have to check all the Markov model’s possible orders.

Probabilistic context-free grammar (PCFG) model is the extension of the Markov model with regular grammar. Each production is assigned with some probability, which is calculated as the multiplication of every production. This grammar also helps to generate word mangling rules [25]. Weir et al. [8] observed that not all guesses have the same probability of cracking a password. They aimed to decrease the likelihood of breaking the password with a limited number of estimates. They created the structure of each leaked password. They calculate the frequency and the probability of the n-gram character of the password. Yazdi et al. [17] have shown the use of context-free grammar for attacking passwords and their defenses. Ma et al. [26] used the probability-threshold graphs, which showed a significant advantage over guess-number graphs. Wang et al. [6] proposed an algorithm that automatically creates a fuzzy probabilistic context-free grammar of the password. Guo et al. [27] proposed a Lightweight Password Strength meter (LPSE). They make a vector representation of the password, assigned scores based on cosine similarity, and edit the distance between the passwords. Combining all the scores, they framed a set of rules that can be easily deployed to the client-side to measure the password’s strength. In this model, when a user tries to register, it takes a small bit of information from the user. The base dictionary of less sensitive passwords trains their fuzzy PCFG model and evaluates the password’s strength. They also suggest a strong password for every weak text password.

Machine learning and deep learning in practice improvised the existing password guessing and generation strategies. These algorithms make use of publicly available leaked data sets and learn to scale and guess the password. Melicher et al. [18] proposed a Recurrent Neural Network (RNN) based model that takes an input and learns to predict the next character in a sequence. They have used character-level language models. The RNN can then generate text character by character that will look like the original training data. In 2016, Yong et al. [28] proposed that neural networks would model human-created English passwords. They trained their model by splitting the password into the n-gram character for predicting the next character. PassGAN [10] is a deep learning method for password generation using Wasserstein GANs (IWGAN) [29] as the model. GAN learns the distribution of the training set and mimics the same distribution taking noise as an input. The main component of the PassGAN is a residual block. They comprise two 1-d convolutional layers, connected by rectified linear units (ReLU) activation functions. Liu et al. [30] proposed a password generator called the GENPass. It uses a combination of probabilistic and neural networks to guess passwords. PCFG divides a password into basic units consisting of characters and numbers. For example, ’password123’ can be divided into two units, ’ L8’ and ’D3’. Neural networks can detect the relationship between characters that PCFG cannot. A password is first encoded into a sequence of units. LSTM model is trained with the preprocessed wordlist. The generated password is sent to a CNN classifier trained with different raw datasets. The classifier predicts which dataset the password most likely comes from. Ciaramella et al. [31] designed a proactive password checking, which self-updates whenever a user tries to choose/change the password using neural networks and statistical techniques. Authors in [32] analyzed the password policies of a hundred websites and proposed an integrated approach for modeling the attackers with multiple layer perceptron (MLP) neural networks. They used the features produced by TF-IDF (term frequency-inverse document frequency) and claim to achieve a classification accuracy of 97.7 with only 10,000 passwords. However, this work does not talk about password modeling and memorability.

The above-discussed schemes state different password strength estimation techniques based on static rules and machine learning models. Rule-based schemes are easily susceptible to dictionary attacks, while machine learning models are prone to adversarial attacks. These models also become cumbersome when it comes to lightweight deployment at the client page. This work tries to mitigate these challenges and proposes a BiGAN based password generator to train an effective ensembled strength meter. We present new password compliances after performing experiments and comparisons based on existing as well as proposed estimators.

3 Preliminaries

3.1 Bi-directional GAN (BiGAN)

A generative model is an unsupervised model, which learns any data distribution to generate new unseen samples. These samples can be used for training intelligent models, testing limitations of existing solutions. GANs lack an efficient inference mechanism, which prevents them from reasoning about data at an abstract level. Bi-directional GAN is an extension of the GAN-like adversarial network by adding a latent generative component. The generator learns the probability distribution of the data and generates new samples by adding noise to it. It aims to fool the discriminator. The additional component, i.e. encoder also learns the data distribution and generates real encoding. The discriminator discriminates both the distributions from the data and latent space. Though these components are not allowed to communicate, they should learn to generate different samples to fool the discriminator.

3.1.1 BiGAN Components

- **Generator (Decoder)** has a prior belief on the latent space z i.e., $_{P_Z(z)}$. Given a draw from this latent space, the generator (G) outputs a synthetic sample i.e., $G(z|\theta_Z) : z \rightarrow x_{synthetic}$.
- **Encoder** works like the inverse of the generator. From a given data space, the encoder (E) outputs a real encoding i.e., $E(x|\theta_E) : x \rightarrow z$. The equation states that given the input x, which is a real image, it outputs real encoding.

– **Discriminator** classifies if a sample given to it is from the real data distribution $P_X(x)$ or the synthetic data distribution $P_G(x|z)$. It also classifies the encoding as real $P_E(z|x)$ or synthetic $P_Z(z)$.

3.1.2 BiGAN Objective Function

BiGAN has the following objective funciton, similar to GAN. It aims to minimize the loss of generator and encoder, while maximize the discriminator's loss to get the most real encoding.

$$\min_{G(z|\theta_G),E(x|\theta_E)} \max_{D(\{x,z\}|\theta_D)} V\big(D\big(\{x,z\} \mid \theta_D\big), G\big(z \mid \theta_G\big), E\big(x \mid \theta_E\big)\big) \tag{1}$$

3.1.3 Advantages over GAN

– GAN generator maps latent samples to generated data. In BiGAN, Encoder helps in reverse mapping and ends up learning arbitrary distributions.
– In GAN, the generator extracts only certain semantic features and updates the parameter. In BiGAN, Encoder assists in extracting all semantic features, which serves as a useful feature representation for learning semantic tasks.
– It has a faster converge rate than GAN as it sends a joint distribution sample to the discriminator and ends up generating similar unseen samples in less time.

3.2 One-Class Support Vector Machine

Let us take a look at the traditional two-class support vector machine(SVM). Consider a data set having data points $(w1, u1), (w2, u2), \dots, (w_n, u_n)$ where w_i is the input point, and u_i indicates the target class. SVM learns to segregate these data points into their respective target labels with maximum inter-class distance [33, 34]. It is used for both classification and regression tasks.

One-class SVM, a variant of SVM, is useful in scenarios when most of the data points belong to the same class. It has been proved quite successful in detecting anomalies, fraudulent activities, document classification [35], and boundary detection [36]. The leaked password datasets and similar generated BiGAN samples comprise to same target label as a weak password. Therefore, it suits to use one-class SVM over the SVM in this context.

3.3 Long Short Term Memory (LSTM)

LSTM is a variant of the recurrent neural network (RNN), capable of learning long-term dependencies. It is good in extracting patterns because it contains memory, which helps memorize the next event on looking at continuous data. We have used it to crack the password. We proposed to split each password into a 3-g character, train, and predicted the next character. We calculate the number of guesses for a

Table 1 Entropy estimation of password policies

Type	Pool of characters
Lower case characters	26
Upper case characters	26
Digits	10
Special case characters	10
Total space	72
Length	**Bits of entropy $log_2(space(72)^{length})$**
10	61.70
11	67.87
12	74.04
13	80.20
14	86.39
15	92.55
16	98.71
Poor passwords	**With high entropy**
Password@123	74.04
User#9000000009	92.55

password to determine the strength of the password. We have kept the threshold at 10^5 for a strong password.

Using weighted ensembling of the strength estimation and guessing time evaluated by one-class SVM and 3-g LSTM, we determine the password's strength. We have trained the one-class SVM model with the combination of leaked passwords and the generated ones by BiGAN. LSTM predicts the time and effort required for guessing the password.

4 Proposed Password Design Techniques

4.1 Issues in Existing Systems

The password guessing success has been defined based on space and entropy. A high entropy makes a password strong and unpredictable. However, it is not a sufficient condition for a strong password. NIST password policies have entropy in the range of 4 to 32. It also achieves entropy as high as 236 with intelligent combinations. Increasing computing power and GPU resources requires a minimum of 50-bit entropy for a semi-secure password. The above-proposed password policies have entropy in the range of 60 to 100. There has been a tradeoff between length, entropy, their secure combinations, and memorability. Passwords with high entropy also suffer offline trawling attacks. Table 1 lists the available keyspaces, entropy, and weak passwords with high entropy. We consider these issues and propose new policies by placing the different keyspaces based on some random function to increase difficulty.

Usually, users choose passwords based on their memory and related information. We publicize our details on social platforms that make the password vulnerable to attack. It is not challenging to generate a strong password as any randomly generated text would be hard to guess. On the other hand, it makes it tough to memorize such passwords. While creating rules and running experiments, we studied several rules used across different sites. They force users to abide by certain constraints, which makes the selection harder and challenging. That restriction policies and pieces of advice make users settle with a compromised and stringent to remember password.

4.2 Proposed Policies

We propose three password design methodologies in Algorithm 1 and further analyze their strength. With the proposed techniques, we add enough randomness that makes it immune to guessing attacks while allowing them to memorize and retrieve the password if the user forgets.

Algorithm 1 Generate Secure and Memorable Passwords

```
 1: procedure GenSecurePassA(phrase, year, dob)
 2:     securePhrase ← turn characters to uppercase at the index places of each digit of
        the year.                                                  ▷ "SeCUre password".
 3:     randSum ← add the year to dob to make it more random.                    ▷ 3998.
 4:     indexSum ← find the sum of the digits of randSum.                           ▷ 2.
 5:     securePhrase ← append each digit of randSum at the index places.
 6:                                                              ▷ "SeCU3re pa8s9sword".
 7:     securePhrase ← update the spaces by choosing the special symbol at indexSum
        and subsequent indexes of set specialSymbols.           ▷ "SeCU3re@pa8s9sword".
 8:     return securePhrase
 9: end procedure

10: procedure GenSecurePassB(phrase, rand, number)
11:     remainderList ← take mode of rand with the length of each word.          ▷ [3, 7].
12:     securePhrase ← turn characters to uppercase for each element in the array
        remainderList.                                             ▷ "secUre passworD".
13:     randSum ← add the rand to number to make it more random.           ▷ 9000019159.
14:     indexSum ← find the sum of the digits of randSum.                           ▷ 3.
15:     securePhrase ← append each digit of randSum at the index places.
16:                                                              ▷ "s0e1cUre5 pas9sworD".
17:     securePhrase ← update the spaces by choosing the special symbol at indexSum
        and subsequent indexes of set specialSymbols.          ▷ "s0e1cUre5#pas9sworD".
18:     return securePhrase
19: end procedure

20: procedure GenSecurePassC(phrase, number, year)
21:     securePhrase ← add the digit to each letter of the phrase.  ▷ "ugewtg rcuuyqtf".
22:     securePhrase ← turn characters to uppercase at the index places of each digit of
        the year.                                                   ▷ "uGewTg RcUuyqtf".
23:     remainder ← calculate mod of every digit in input year by number.        ▷ 1101.
24:     indexSum ← find the sum of the digits of remainder.                         ▷ 3.
25:     securePhrase ← append each digit of remainder at the index places of each digit
        of the year.                                             ▷ "uG1ewT0g R1cU1uyqtf".
26:     securePhrase ← update the spaces by choosing the special symbol at indexSum
        and subsequent indexes of set specialSymbols.           ▷ "uG1ewT0g-R1cU1uyqtf".
27:     return securePhrase
28: end procedure

29: specialSymbols ← [!,*,@,-,$,%,_,#,(,)]     ▷ users may define their own set of symbols.
30: phrase ← choose a phrase of min length 10.                      ▷ "secure password".
31: S ← choose password generation function.
32: switch S do
33:     case A
34:         year ← choose any year of importance.                 ▷ let user inputs 2003.
35:         dob ← enter your date of birth.                       ▷ let user inputs 1995.
36:         securePhrase ← GenSecurePassA(phrase, year, dob)
37:     case B
38:         rand ← choose any random number.                       ▷ let user inputs 159.
39:         number ← choose any large number.               ▷ let user inputs 9000019000.
40:         securePhrase ← GenSecurePassB(phrase, rand, number)
41:     case C
42:         number ← enter a number between 1 to 10.                 ▷ let user inputs 2.
43:         year ← choose any year of importance.                 ▷ let user inputs 1947.
44:         securePhrase ← GenSecurePassC(phrase, number, year)
```

There are some universal principles and compliances of keeping passwords long and a mix of different symbols. Users fail to memorize due to a more complex design. The proposed methods take care of maintaining inputs memorable and straightforward. It helps us keep the plain text in memory and use any transcription technique to convert into a strong password. As the methods are public, we add enough randomness to add strength to our passwords. In most cases, inputs for passwords are personal information like name, organization, self-owned accessories, etc., which makes them essentials for a memorable password. Thus, the proposed methods take the same inputs from the user's memory and parse them in a function by adding a fair amount of randomness. It makes the generated passwords both memorable and reasonably secure. These policies are also designed to be safe against targeted attacks. There needs to be enough randomness to resist targeted attacks in the age of social exposure and availability of personally identifiable information (PII) across social sites. The most vulnerable PII are name and phone, followed by the date of birth and interests. There arises a question of usage and memorability again. So, the above-proposed rules mingle all the requirements and test them with severe guessing attacks.

The proposed methods have been tested against various meters on different scales. It has been evaluated based on password length, compliance, PARS, zxcvbn, LPSE, and our proposed strength estimator. We generated around two hundred responses for each policy.

5 Security Model

This work focuses on designing secure and memorable passwords. The system comprises a social site or any cloud service provider where a user registers. There is a malicious site where the user is also associated. There is a natural tendency to use sister passwords across different sites, which leaks much sensitive information. A computing server is configured to run all-out guessing attacks using the proposed generation and guessing algorithms. Both the adversary and service providers configure the server based on their capabilities.

An adversary can gather information by establishing various channels. He can gather level I (more sensitive information) on a private channel. Level I personal identifiable information (PII) generally comprises private information like hobbies, interests, educational qualifications, etc. The adversary tries to collect the Level II PII from social sites, cloud providers, and malicious servers. Level II information consists of publicly available data like name, date of birth, and so. Adversary frames targeted offline attacks on individuals to breach their security. The computing server consists of efficient BiGAN architecture, which generates multiple possible passwords. An LSTM framework calculates the number of guesses required to guess in a locking or throttling server.

The computing server uses the proposed password policies and suggests strong and memorable passwords. It uses all the factors like entropy, the chances of breaking the password based on the available keyspace, and proper combination to defy the dictionary attacks. For simulating the above-discussed scenarios, we have gathered information from different users and ran experiments to evaluate and suggest secure passwords based on different metrics. Fig. 1 illustrates the relationship between all the actors and entities discussed above.

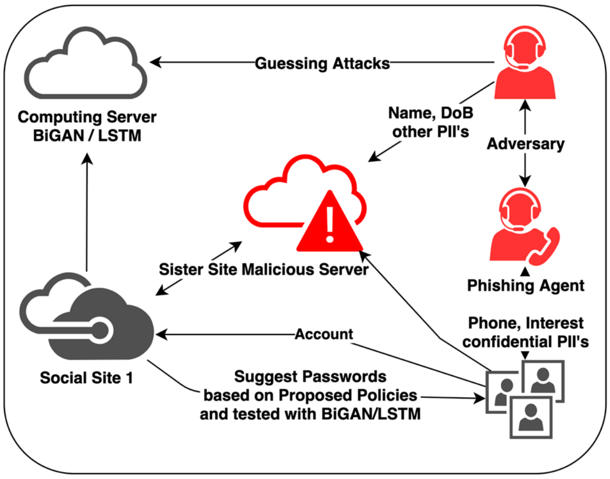

Guessing Attacks
Computing Server BiGAN / LSTM
Name, DoB other PII's
Adversary
Phishing Agent
Sister Site Malicious Server
Social Site 1
Account
Phone, Interest confidential PII's
Suggest Passwords based on Proposed Policies and tested with BiGAN/LSTM

Fig. 1 Security model

6 Proposed Password Generation and Estimation Framework

The proposed password strength estimation technique learns to estimate the strength of user passwords. It evaluates the strength based on the prediction by a weighted average of one-class SVM and tuned 3-g LSTM. These models have been trained on the adversarial passwords generated by a tuned BiGAN. It helped to increase the accuracy, adding more vulnerable passwords to the training list. It is the very first approach that combines generation, strength estimation strategies, and guessing attacks on a password to the best of our knowledge.

In the first phase, the framework collects the leaked dataset as input. After pre-processing cleaning and conversion steps on the input dataset, it is fed to the BiGAN generator and encoder. The discriminator performs intensive training to generate real adversarial samples. The input dataset and the generated data are taken as input for training by one-class SVM and LSTM in the next phase. One-class SVM model learns to classify the password strength while LSTM predicts the efforts for guessing the password. Based on both these strengths and prediction scores, the password is categorized as strong, medium, or weak. The major components of this framework are shown in Fig. 2 and described subsequently.

6.1 Data Collection, Cleaning and Conversion

We have used two publicly available leaked password datasets, i.e., RockYou dataset[3] and MySpace dataset.[4] RockYou dataset contains 14,341,564 unique passwords from 32,603,388 accounts. All the RockYou passwords are unencrypted and stored

[3] https://www.kaggle.com/wjburns/common-password-list-rockyoutxt.

[4] http://downloads.skullsecurity.org/passwords/myspace.txt.bz2.

in plain text. My Space dataset contains 27,484,128 unique passwords from a total of 60,213,024 distinct emails.

Many passwords in the data set are not hashed. There exists a repetition of passwords that needs to be removed. They contain non-ASCII characters. We removed the non-ASCII characters from the data set. Many passwords are of only four characters and do not fit for a strong password. We have taken the minimum password of length six and a maximum of fourteen characters.

We split the data set as training and testing in the ratio of 80:20. We take the training data and convert it into an image of shape seven x max size of the password. We set the maximum size as fourteen by either padding space or truncating the extra to make all the equal dimension input. We have considered all the digits (0–9), upper case alphabets (A, B, …, Z), lower case alphabets (a, b,c, …, z), and special characters (@, +, −, & …) a total of ninety-two including space. Every character is assigned a number by taking its decimal representation and converting it into a seven-bit binary representation.

6.2 Training

Every single password in the training set is represented as a seven-bit binary representation and represented as an image, an input for BiGAN. The image's size is (seven x n), where n is the maximum password length. Seven bits are used for representation as we have ninety-two unique characters stated in the previous subsection. For one-class SVM, we have extracted the feature as the password's length, the number of uppercase and lowercase characters, the number of special characters, and the number of digits.

In the proposed framework, BiGAN is trained in the first phase, and their generated samples become the input for training in the next phase. In the second phase, one-class SVM and LSTM models are trained. Their weighted score is used to evaluate the strength of the password.

6.2.1 BiGAN Training

The leaked password samples become an input to the generator and encoder of the BiGAN setup. The samples are preprocessed first and then converted into an image with a dimension of seven x maximum size of the password, i.e., fourteen in our case. We transformed each password into an image and fed it to the discriminator (D). The generated samples of encoder (E) and generator (G) become an input to the discriminator D. It generates a picture with a dimension of seven x fourteen as output. The discriminator predicts the samples as real and fake, and the prediction value ranges between 0 and 1. The samples less than equal 0.5 are represented as 0, and the rest as 1. Then, we take seven-bit sequentially and convert the binary representation to decimal. We do the reverse of character to binary mapping as we need the character back. Using character to decimal representation, we turn the decimal back to its character representation. All the passwords will be fourteen characters

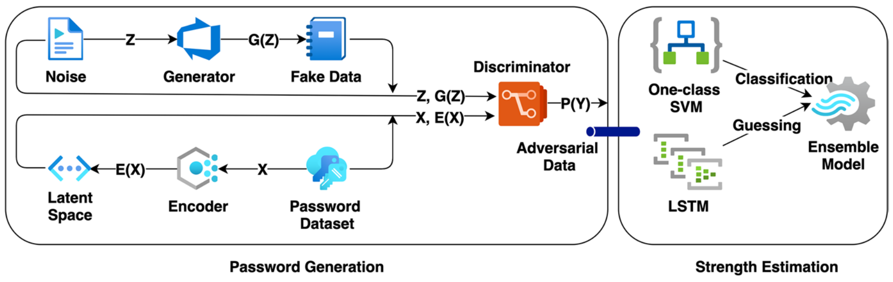


Fig. 2 Proposed framework

as we had added space padding. After generating and converting it into a password, we remove the padded space from the password. With a faster convergence rate of BiGAN, we successfully generated new unseen samples of passwords.

6.2.2 Ensemble Training

In one-class SVM, 80% of the RockYou dataset and MySpace dataset with BiGAN generated samples have been used for training. It trains on features like the length of the password, the number of uppercase and lowercase characters, the number of digits, and the number of special characters. For labeling the training set, password strength estimator PARS[5] has been used. It gives a rough estimate of the password strength. Using one-class SVM is influenced by the fact that all the leaked passwords should be categorized as weak. PARS evaluation metrics have been scaled down to fit this observation. LSTM is trained with the same dataset to measure the efforts required to guess the password. Each sample is split into three character grams to predict the next character. The number of passwords generated for guessing tells about the password's strength. Observing the results, we fixed 10^5 guessing attempts as a threshold for a weak password and 10^{10} as a medium. By doing the weighted average of both these models, the strength of the password is estimated.

7 Experiment Setup and Configuration

7.1 Setup

Docker setup with an ubuntu image has been used to run these experiments. It helps in easy installation and migration. We configured a host machine with processor Intel®Core$_{TM}$ i7-7700 CPU @ 3.60GHz 8, and Memory of 8 GB. It takes nearly a day to run BiGAN architecture with one lakh epochs. We have used the same setup for running our ensemble model of one-class SVM and 3-g LSTM model. It took around five hours for training.

[5] http://www.pars.gatech.edu/.

7.2 Model Configuration

This section lists all the hyperparameters required for training the BiGAN, one-class SVM, and LSTM.

7.2.1 BiGAN Architecture

- **Batch size** indicates the number of data points as input in BiGAN at each step. The batch size is chosen to be 64.
- **Number of iterations** tells about the number of iterations in BiGAN training. We ran 199,000 iterations to compare the results with PASSGAN, but we improved results in 100,000 iterations.
- **Hidden Layers:** Our BIPASSGAN model consists of Generator (*G*), Discriminator (*D*), Encoder (*E*), each consists of two hidden layers with 512 units each. First, the hidden layer is followed by nonlinearity. The second layer is followed by batch normalization and nonlinearity. It becomes the input to a linear prediction layer.
- **Output sequence length** is the number of characters in the strings generated by the generator (*G*). We have kept the maximum length of the password as 12 characters.
- **Input noise vector size (seed)** indicates how many random numbers from a normal distribution are fed as input to generate samples. We set this size to 128 floating-point numbers.
- **Maximum number of examples** represents the maximum number of the passwords generated by our BIPASSGAN. We have set the maximum number of examples as 1,000,000.
- **Nonlinearity:** In discriminator (*D*) and encoder (*E*), leaky Relu with a leak of 0.2 is used. In generator (*G*), a standard Relu is used.
- **Adam optimizer's hyper-parameters:**
  - Learning rate (α), i.e., how the curve goes to optimizes the loss. It has been set as 20^{-5}.
  - Coefficient ($\beta 1$), specifies the decay rate of the running average of the gradient. It has been set as 0.5.
  - Coefficient ($\beta 2$), specifies the decay rate of the gradient's square's running average. It has been set as 0.9.

7.2.2 One-Class SVM Architecture

- **The size of the input** is the number of passwords available. We have 1,000,000 passwords as input.
- **Kernel:** The function of the kernel is to take data as input and transform it into the required form. We have used the default RBF(Radial Basis Function).

- **Gamma**(γ): It defines the influence of a single training example. We have chosen the default value auto.
- **Nu**(ν): The parameter nu is an upper bound on the fraction of margin errors and a lower bound on the fraction of support vectors relative to the total number of training examples. It is set to be 0.005.

7.2.3 LSTM Architecture

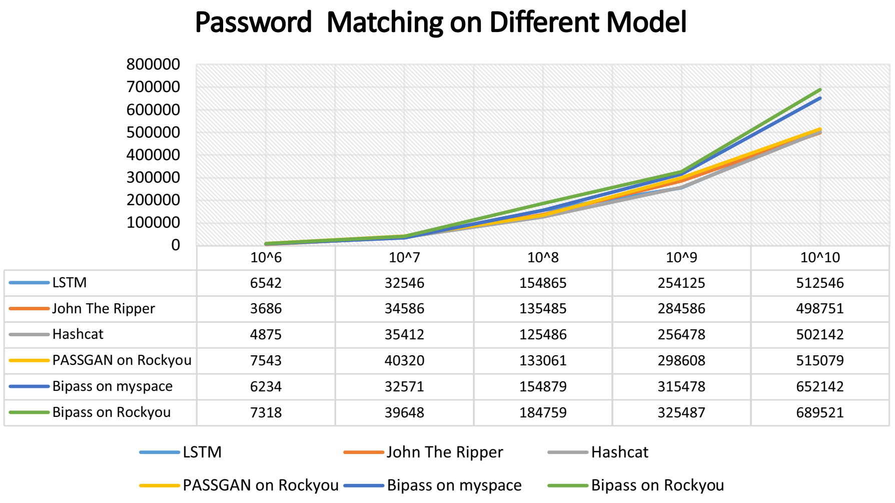
- **Order:** The order of LSTM has been set to 3.
- **Step length:** The step length we take to get samples from the corpus. It has been set as 1.
- **Epochs:** Number of times we train on our complete data is set as 190,000.
- **Batch size:** The size of data samples in each training step. It is set as 32.
- **Latent dim:** Dimension of the LSTM network has been set as 64.
- **Dropout rate:** Regularization with dropout is set 0.2.
- **Gen amount:** It is the number of passwords generated for guessing. We generate 1000 passwords in every iteration to guess the input password.

8 Result

This section covers password generation results using BiGAN, password strength evaluation using one-class SVM, and password guessing using 3-g LSTM. It also lists comparison based on different existing schemes [10, 17, 18, 27]. It describes the results of experiments run over the proposed password policies and compares them with the existing policies [23].

8.1 BiGAN Results

Training with BiGAN shows significant improvement in time and cost, maintaining the equivalent performance of password generation. Figure 3 illustrates the comparison of the proposed technique BIPASSGAN and LSTM with PASSGAN, John the Ripper, and Hashcat. The horizontal axis represents the number of passwords generated by different techniques on different datasets. The vertical axis shows the number of password matches. The proposed techniques show the best matching rate compared to the existing models. Training with leaked datasets achieved 87% accuracy, as it was labeling similar to the existing password meters. It was not able to classify many of the weak passwords. Adding the BiGAN generated samples dropped the accuracy and strength of strong passwords to 52%. It ranks more weak passwords, which were initially misclassified as secure by the raw model or other

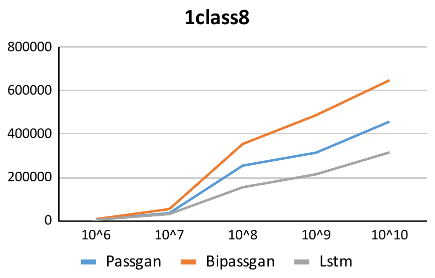


	10^6	10^7	10^8	10^9	10^10
LSTM	6542	32546	154865	254125	512546
John The Ripper	3686	34586	135485	284586	498751
Hashcat	4875	35412	125486	256478	502142
PASSGAN on Rockyou	7543	40320	133061	298608	515079
Bipass on myspace	6234	32571	154879	315478	652142
Bipass on Rockyou	7318	39648	184759	325487	689521

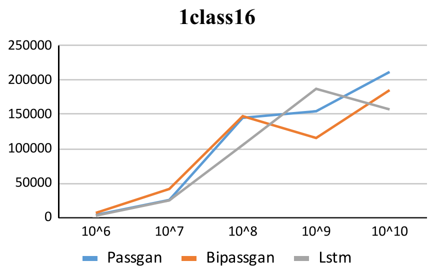
Fig. 3 Password generation results

strength estimators. It generated meaningful passwords considering different keyspaces, higher entropy, and the use of personally identifiable information (PII).

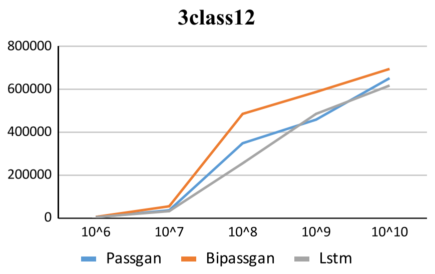
We have also checked BiGAN capabilities on different password policies, i.e., 1class8, 1class16, 3class12, and 4class8. They are based on different password lengths and classes, indicating the composition of upper-case letters, lowercase letters, numbers, and symbols [18]. Figure 4a illustrates 1class8 password policy results with passwords of minimum length 8. It achieves a high matching rate of up to 70,000 using BiGAN due to less space and low entropy. Figure 4b illustrates the 1class16 password policy results, which has a password of minimum length 16. It is where our guessing suffers some loss due to larger space and higher entropy. BiGAN also suffers from generating meaningful passwords of 16 characters. Our proposed model has a limitation of generating matching passwords up to 12 characters. This work can be extended to produce more realistic results using multi-task GAN fed with user PII. Figure 4c illustrates 3class12 password policy with password of minimum length 12. It should have any of the three uppercase, lowercase alphabet, or numeric symbols. It achieved the best results even with larger keyspace and higher entropy. Both BiGAN and LSTM faired well and outperformed PASSGAN with less computing effort. Figure 4d illustrates a 4class8 password policy with a password of minimum length 8. It should have all of the uppercase, lowercase alphabet, and numeric symbols. Passwords of length eight characters are more standard ones and framed with tested policies. A moderate entropy but complex design led to optimum but not the desired performance. However, overall these results show that BiGAN could generate more unique passwords than PASSGAN, while our tuned LSTM architecture also faired well.



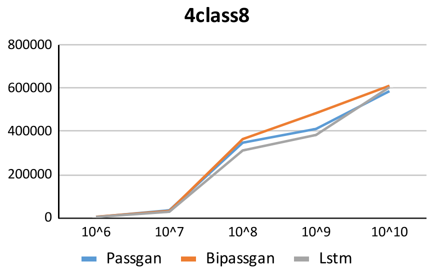
(a) 1class8 Password Policy



(b) 1class16 Password Policy



(c) 3class12 Password Policy



(d) 4class8 Password Policy

Fig. 4 Comparison based on different password policies

8.2 Password Collection, Estimation, and Suggestion

Every user puts his randomness in choosing the password. This randomness revolves around his sensitive information about what he thinks is private. With privacy becoming a myth and confidential information getting leaked, it has become easier to crack someone's password. We ran another experiment asking users to fill in strong passwords. We also asked for personally identifiable information (PII) like name, dob, and phone number. We tested those passwords with our proposed model, which classified many of them as weak or moderate.

Table 2 lists a few of the surveyed passwords with their corresponding strengths evaluated by existing password meters and the proposed model. These passwords were evaluated against different password meters (Pars, zxcvbn, LPSE, and our proposed model). It can be observed that there is a significant difference between the evaluations. With most user passwords classified as weak, we suggested a strong password generating new unseen samples using BiGAN and testing its strength by our ensemble method.

We have tested our Ensemble model with 20% of RockYou and MySpace leaked password datasets. The result illustrates that 11,983 (16%) of the passwords were classified as strong, 23,257 (31%) as a medium, and the remaining as weak (52%). It can be inferred from these comparisons that our proposed strength estimator is guessing most of the vulnerable passwords as weak compared to the existing ones.

8.3 Password Design Methods Evaluation

The passwords based on the proposed design methods in Sect. 4 were generated by taking personal user inputs and adding a fair amount of randomness over that. We designed scripts for every method and successfully created around two hundred passwords in each category. We analyzed each of the methods on various meters and different scales. These methods have also been compared with the methods proposed by authors in [23]. Figure 5a compares the result based on the scale of length. It clearly illustrates that the proposed methods (one, two, and three) edge over the existing methods. Figure 5b draws a further comparison based on the compliance of the methods (four, five, and six) and successfully generates stronger passwords. Figure 5c and d evaluates the methods based on existing and proposed metres. The result illustrates a better performance of the proposed algorithms.

The proposed policies consider entropy one of the strengths but not the sufficient condition for a secure password. The rules mingle the key spaces integrating PII to make memorable passwords. A large sample of generated passwords is then tested for guessing attacks. The attacks are carried in various scenarios like locking and throttling service providers' environments, SSH login, or even relaxing those assumptions. It easily passes both the level 1 and 2 criteria of password certification. Level 1 demands a minimum of 1024 attempts, while level 2 asks for 16,284 attempts. With high processing GPU available, these numbers are not enough. We also considered another attack scenario of a 50% leak, where the attacker has access to half of the keyspace. However, the generated passwords using the proposed policies require significantly more severe attempts.

9 Conclusion and Future Work

This paper presented a password monitor with modeling, generation, and estimation frameworks. The password generation techniques use a bi-directional generative adversarial network (BiGAN), which successfully generated an equal number of new unseen samples in 1,00,000 epochs, nearly half computationally intensive to the existing techniques. We also listed some password creation policies based on functional patterns, high entropy by adding a fair amount of randomness. The proposed methods allow users to take personally identifiable information for designing memorable passwords. These policies helped to gain a 5% to 10% surge in creating strong passwords. A password strength estimation scheme based on the weighted evaluation by a one-class SVM and tuned LSTM has been proposed. It successfully classified the most vulnerable passwords as weak and marked only 16% of the test set as strong passwords. The proposed BiGAN technique can be extended as a multi-task GAN with more inputs to the discriminator, aiming for targeted classification and generation.

Acknowledgements We acknowledge the Ministry of Human Resource Development, Government of India, for providing fellowship to complete this work.

Table 2 Password-strength estimation

Password	Pars	zxcvbn	LPSE	Our model
$ANju8820	Very strong	Strong	Strong	Weak
Kundan@#1994	Very strong	Medium	Medium	Good
JAY!@#882037820	Very strong	Strong	Medium	Good
Bhavendra206	Strong	Good	Weak	Weak
8963343643	Weak	Weak	Medium	Medium
25081998@devil	Strong	Good	Strong	Weak
lucy@!#9863	Strong	Weak	Weak	Weak
Sirdhar123456!@	Strong	Good	Medium	Medium
lovemysoul	Weak	Weak	Weak	Weak
liveThelife@1231224	Strong	Strong	Medium	Medium
netdome8752dxja	Strong	Good	Medium	Strong
amazon2345da	Good	Good	Medium	Medium
SANjgaydyatig	Good	Strong	Medium	Strong
Riya@199578fdjy	Strong	Good	Strong	Weak
juhiGyani@kichkich	Strong	Strong	Medium	Weak
Shikhatek+457674	Strong	Good	Strong	Medium
JyotiKhateN1235	Strong	Good	Medium	Weak
PiyaDon45678	Strong	Medium	Medium	Weak
RiyaJay14758788534	Strong	Strong	Medium	Medium
123456789	Weak	Weak	Weak	Weak
987654321	Weak	Weak	Weak	Weak
am1AStrongPassword#	Very strong	Strong	Strong	Weak
11@Mushtehara	Very strong	good	Medium	Strong
@nebullae	Very strong	Weak	Weak	Medium
!ThereisnoHope12!	Strong	Strong	Strong	Strong
Creator@92	Strong	Weak	Weak	Medium
ds101@iitpnitrkl	Very strong	Strong	Medium	Strong
Xiaomi Redmi Airdots	Strong	Medium	Medium	Medium
joinstar1Q!	Strong	Medium	Strong	Medium
Indianlovesindia@9371	Very strong	Strong	Strong	Strong
@Enjoy321#	Very strong	Weak	Medium	Medium

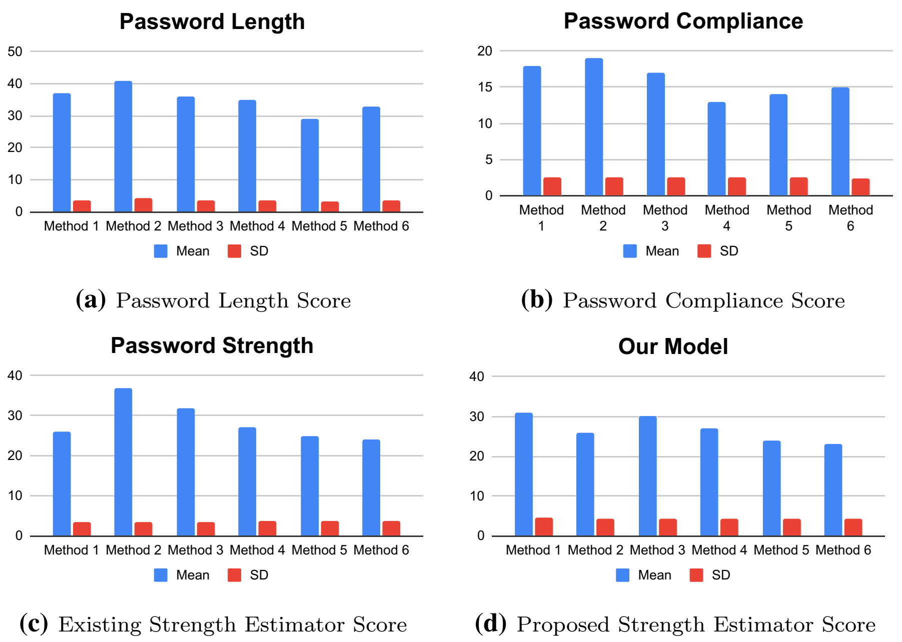
Funding Not applicable.

Data Availability Not applicable.

Code Availability Not applicable.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.



(a) Password Length Score **(b)** Password Compliance Score

(c) Existing Strength Estimator Score **(d)** Proposed Strength Estimator Score

Fig. 5 Password strength estimation

References

1. Kim, H., Huh, J.H.: Pin selection policies: are they really effective? Comput. Secur. **31**(4), 484–496 (2012)
2. Nandakumar, K., Nagar, A., Jain, A.K.: Hardening fingerprint fuzzy vault using password. In: Proceedings of the International Conference on Biometrics, pp. 927–937. Springer (2007)
3. Galterio, M.G., Shavit, S.A., Hayajneh, T.: A review of facial biometrics security for smart devices. Computers **7**(3), 37 (2018)
4. Wang, D., Cheng, H., Wang, P., Huang, X., Jian, G.: Zipf's law in passwords. IEEE Trans. Inf. Forensics Secur. **12**(11), 2776–2791 (2017)
5. Wang, D., Wang, P.: The emperor's new password creation policies. In: Proceedings of the European Symposium on Research in Computer Security, pp. 456–477. Springer (2015)
6. Wang, D., He, D., Cheng, H., Wang, P.: fuzzypsm: a new password strength meter using fuzzy probabilistic context-free grammars. In: Proceedings of the 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 595–606 (2016)
7. Doucek, P., Pavlíek, L., Sedláček, J., Nedomová, L.: Adaptation of password strength estimators to a non-English environment-the CZech experience. Comput. Secur. **101757**, 02 (2020)
8. Weir, M., Aggarwal, S., Medeiros, B., Glodek, B.: Password cracking using probabilistic context-free grammars. In: Proceedings of the 2009 30th IEEE Symposium on Security and Privacy, pp. 391–405 (2009)
9. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. Adv. Neural Inf. Process. Syst. **27** (2014)
10. Hitaj, B., Gasti, P., Ateniese, G., Pérez-Cruz, F.: Passgan: a deep learning approach for password guessing. CoRR, abs/1709.00440 (2017)
11. Donahue, J., Krähenbühl, P., and Darrell, T.: Adversarial feature learning. CoRR, abs/1605.09782 (2016)
12. Weir, M., Aggarwal, S., Collins, M., Stern, H.: Testing metrics for password creation policies by attacking large sets of revealed passwords. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, pp. 162–175 (2010)

13. Wang, D., Zhang, Z., Wang, P., Yan, J., Huang, X.: Targeted online password guessing: An underestimated threat. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1242–1254 (2016)
14. Wang, D., Wang, P., He, D., Tian, Y.: Birthday, name and bifacial-security: understanding passwords of Chinese web users. In: Proceedings of the 28th USENIX Security Symposium (USENIX Security 19), pp. 1537–1555 (2019)
15. Florêncio, D., Herley, C., and Van Oorschot, P.C .: An administrator's guide to internet password research. In: Proceedings of the 28th Large Installation System Administration Conference (LISA14), pp. 44–61 (2014)
16. Van Heerden, R.P., Vorster, J.S.: Using Markov models to crack passwords. In: Proceedings of the 3rd International Conference on Information Warfare and Security: Peter Kiewit Institute, University of Nebraska, Omaha, USA, pp. 24–25 (2008)
17. Yazdi, S.H.: Probabilistic context-free grammar based password cracking: attack, defense and applications. Comput. Sci. (2015)
18. 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: Proceedings of the 25th USENIX Security Symposium (USENIX Security 16), pp. 175–191, USENIX Association, Austin, TX (2016)
19. Bošnjak, L., Sreš, J., Brumen, B.: Brute-force and dictionary attack on hashed real-world passwords. In: Proceedings of the 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1161–1166 (018)
20. Zoebisch, F., Vielhauer, C.: A test tool to support brute-force online and offline signature forgery tests on mobile devices. In: 2003 International Conference on Multimedia and Expo. ICME '03. Proceedings (Cat. No. 03TH8698), vol. 3, pp. III–225 (2003)
21. Wang, D., Wang, P.: Offline dictionary attack on password authentication schemes using smart cards. In: Desmedt, Y. (ed.) Information Security, pp. 221–237. Springer International Publishing, Cham (2015)
22. Monrose, F., Reiter, M.K., Wetzel, S.: Password hardening based on keystroke dynamics. In: Proceedings of the CCS '99 (1999)
23. Yıldırım, M., Mackie, I.: Encouraging users to improve password security and memorability. Int. J. Inf. Secur. **18**(6), 741–759 (2019)
24. Woods, N., Siponen, M.: Improving password memorability, while not inconveniencing the user. Int. J. Hum. Comput. Stud. **128**, 61–71 (2019)
25. Houshmand, S., Aggarwal, S., Flood, R.: Next gen PCFG password cracking. IEEE Trans. Inf. Forensics Secur. **10**(8), 1776–1791 (2015)
26. Ma, J., Yang, W., Luo, M., Li, N.: A study of probabilistic password models. In: Proceedings of the 2014 IEEE Symposium on Security and Privacy, pp. 689–704 (2014)
27. Guo, Y., Zhang, Z.: LPSE: lightweight password-strength estimation for password meters. Comput. Secur. **73**, 507–518 (2018)
28. Fang, Y., Liu, K., Jing, F., Zuo, Z.: Password guessing based on semantic analysis and neural networks. In: Zhang, H., Zhao, B., Yan, F. (eds.) Trusted Computing and Information Security, pp. 84–98. Springer, Singapore (2019)
29. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein gan. arXiv preprintarXiv:1701.07875 (2017)
30. Liu, Y., Xia, Z., Yi, P., Yao, Y., Xie, T., Wang, W., and Zhu, T.: Genpass: a general deep learning model for password guessing with PCFG rules and adversarial generation. In: Proceedings of the 2018 IEEE International Conference on Communications (ICC), pp. 1–6 (2018)
31. Ciaramella, A., D'Arco, P., De Santis, A., Galdi, C., Tagliaferri, R.: Neural network techniques for proactive password checking. IEEE Trans. Depend. Secure Comput. **3**(4), 327–339 (2006)
32. He, Y., Alem, E.E., Wang, W.: Hybritus: a password strength checker by ensemble learning from the query feedbacks of websites. Front. Comput. Sci. **14**(3), 1–14 (2020)
33. Junli, C., Licheng, J.: Classification mechanism of support vector machines. In: WCC 2000–ICSP 2000. 2000 5th International Conference on Signal Processing Proceedings. 16th World Computer Congress 2000, vol. 3, pp. 1556–1559 (2000)
34. Jamuna, K.S., Karpagavalli, S., Vijaya, M.S.: A novel approach for password strength analysis through support vector machine. Int. J. Recent Trends Eng. **2**(1), 79 (2009)
35. Zhu, F., Ye, N., Wei, Yu., Sheng, X., Li, G.: Boundary detection and sample reduction for one-class support vector machines. Neurocomputing **123**, 166–173 (2014)

36. Manevitz, L.M., Yousef, M.: One-class svms for document classification. J. Mach. Learn. Res. **2**, 139–154 (2001)

Authors and Affiliations

Sanjay Murmu[1] · Harsh Kasyap[1] · Somanath Tripathy[1]

Sanjay Murmu
1811mc11@iitp.ac.in

Somanath Tripathy
som@iitp.ac.in

[1] Department of Computer Science and Engineering, Indian Institute of Technology Patna, Patna, India