Adversarial Password Cracking

Thejas S
Dept of Computer Science
PES University
Banagalore, India
s.thejas@gmail.com

Vinayaka M Hegde
Dept of Computer Science
PES University
Banagalore, India
vinuvinayaka2000@gmail.com

Prof. Sushma E
Dept of Computer Science
PES University
Banagalore, India
sushmae@pes.edu

Prof. Prasad B. Honnavalli
Dept of Computer Science
PES University
Banagalore, India
prasadhb@pes.edu

***Abstract*—Password cracking and exploitation of user data is one of the biggest issues in the domain of cyber security. Generative adversarial networks are a type of machine learning models which are used to generate new data points based on the data which they are trained on. Since it is unsupervised, it identifies and learns patterns if any present in data. It consists of a Generator that tries to generate guesses by learning a password distribution and a discriminator that can discriminate between real and fake passwords generated by the generator. This paper uses this model to generate passwords. There are many existing techniques to generate password guesses which include brute force, rule based generation, other generative models like VAE. While rule-based techniques like HashCat, John The Ripper and others can crack billions of passwords per second, creating rules for these techniques requires expertise and is time consuming. Also, recent research shows that combining deep learning models with these tools can produce significantly better results. This model will be able to learn the hidden patterns present in distribution of the input dataset automatically and thus is easier to use. Also, the model does not require a-priori knowledge of the dataset. In addition, the discriminator once trained can also be used to determine the strength of the password. Finally, accuracy of the implemented model is evaluated by comparing it with an existing password cracker like Hashcat.**

I. Introduction

Passwords are being used for authentication purposes for a long time. It is easy and convenient for implementing and using it. Companies need to make sure that the user's passwords are not compromised. If compromised it can be misused which affects the user's account which may lead to loss of money, comprise identity, etc. In the past there were many attempts to hack the databases and has also happened which is concerning. The passwords are stored in hashed form in the database. If the passwords are strong enough, it will be very difficult to crack it. There are numerous ways like dictionary attack, rule based attack, etc to crack the passwords. These methods require expertise and are time consuming tasks. The output space is also limited.

Generative adversarial networks can be used to generate passwords that humans are likely to use. A model is trained based on a list of real passwords. This data is available as there instances in which the compromised passwords were put online. These GAN's without the domain knowledge will be able to understand how passwords are created and once properly trained, they will generate passwords which are likely used. This approach does not require for us to input any rules, only a list of passwords are required to train the model.
Well trained models can be used to substitute or used in addition to the existing tools for password cracking. Certain patterns in passwords which might not be easy to identify by humans can be understood by this model. The output space of this model is not limited and can be used to generate many passwords. This model can be used as a security measure where the user can check for easily crackable passwords. It can find its applications in checking password strength.

II. Literature Survey

In this section, previous research and results obtained in the field of the project are analysed, by taking into account the various parameters and extent of the project.

Nepal Adversarial [1] is a generative adversarial network trained with a generator trying to generate guesses to break a password scheme. The author has later compared the strength of this adversarial model with rule based methods such as hashcat . Later, the passwords cracked by PassGAN are compared with results of zxcvbn, to determine if the passwords cracked by PassGAN were actually hard to crack. The author suggests using a machine learning based approach for password guessing which is called PassGAN. PassGAN is a generative adversarial network where the generator tries to generate a fake password to fool the discriminator and discriminator tries to distinguish real passwords from fake passwords (passwords generated by generator). They both end up playing a minimax game and optimize the value function. The author has trained the PassGAN model on two datasets - one is RockYou and second is AshleyMadison dataset. Further, an extended version of RockYou dataset was created by including slightly modified variations of the passwords, to represent a more realistic set of passwords. The network was trained upto 70k iterations and a plot of training loss versus iterations is represented by Figure 1.

After training, the set of passwords cracked using PassGAN was evaluated using zxcvbn and the results for RockYou and extended RockYou dataset are summarized in Table 1. From this table, it can be inferred that the average number of guesses for extended RockYou is high since it is a much randomized version of rockYou dataset.

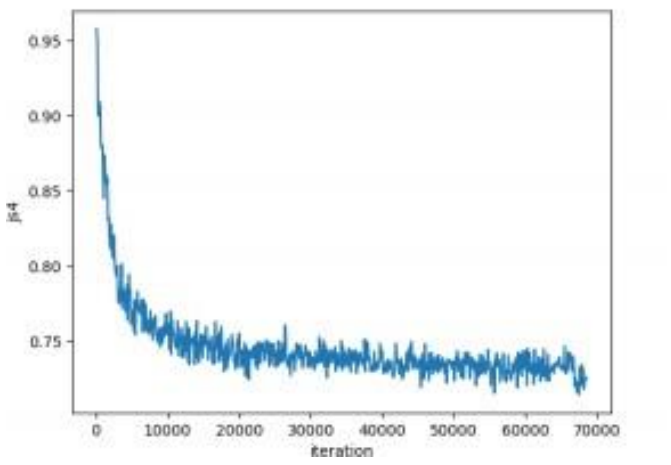

Fig. 1. Training loss v/s number of iterations

A Chen [2] argues that GANs generate high quality password guesses, and can match 51 to 73 percent more passwords than the traditional rule based methods. This shows that deep learning methods can be used for password cracking. The paper begins by stating how carelessness of users can be exploited by malicious hackers in this online world. Next, the paper presents us with various kinds of attacks used by state of the art password guessing tools such as hashcat. Brute force attack generates passwords by going through a combination of characters upto a given length. Dictionary based attacks use a list of words from a list of commonly used passwords that are revealed from the past leaks, and then hash them for further use. Then, it checks these hashes against the unknown hash to crack the unknown hash. Dictionary attacks often succeed mainly because people often choose ordinary words or some common passwords with little variants that can easily be cracked. Rule based attacks use a set of rules on top of the dictionary list.

For example, there could be a rule for concatenating all passwords with some numbers since these kinds of passwords are highly probable. The paper evaluated the performance of PassGAN by running the model on different datasets. The results prove that PassGAN can match 51 to 73 percent more passwords than the traditional rule based methods. Additionally, it was found that the best overall performance was achieved when PassGAN was combined with other techniques, specifically when PassGAN and HashCat were used jointly. This concludes that PassGAN was able to generate real-like passwords and when hashcat was used on these passwords, it was able to perform much better since the password list generated by PassGAN consisted of highly probable passwords.

The author concludes by stating that PassGAN can be effectively used to generate passwords that resemble real passwords since it can extract properties of passwords which could be indistinguishable to human observation. Also, PassGAN can be used without a priori knowledge of the structure of passwords. Also, PassGAN can generate passwords indefinitely whereas other tools can generate a limited number of passwords. The best overall performance was achieved when PassGAN was combined with other rule based techniques.

B Hitaj [3] mainly talks about how deep learning models like PassGAN can be applied to the password guessing problem. B Hitaj starts by stating various kinds of rule based methods available for cracking passwords and later compares the results of using these methods with respect to that of PassGAN. Moving on to the architecture of PassGAN, this paper uses the Wassertian GANs (IWGANs) with Adam optimizer. Various hyperparameters like batch size, number of iterations, output sequence length, size of the input vector have been stated. IWGAN is implemented using ResNets since ResNets continuously reduce the training error as the number of layers increases. Residual Blocks are composed of two 1-dimensional convolutional layers, connected with one another with rectified linear units (ReLU) activation functions. PassGAN was tested on the classic RockYou dataset and linkedin leaked dataset. Figure (2) represents the number of unique passwords generated by PassGAN trained using the RockYou dataset. From the figure, it can be inferred that the number of passwords matched increased until 199,000 iterations. And after that, increasing the number of iterations reduced the number of passwords matched which implies the model started to overfit. So, after 199,000 iterations, the training was stopped.

When PassGAN was compared with hashcat, the results showed that PassGAN performed better and it was observed that PassGAN was able to match more passwords than hashcat. The paper concludes by stating major advantages of PassGAN over rule based approaches and in which situations rule based methods are preferred over PassGAN. It has been observed that rule based methods were able to outperform when the number of guesses allowed to crack a password was small. However, the main downside of rule-based password guessing method is that rules can generate only a small set and finite number of passwords. But, PassGAN was able to eventually generate a large number of passwords. As a result, the best password guessing strategy is to use multiple tools since it was observed that by integrating the output of Best64 HashCat algorithm with the output of the PassGAN, 50.8 percent more passwords could be matched.

V Garg [4] introduces various kinds of data breaches and the possible causes for the same. Then, it investigates how well the PassGAN model performs when trained on a large dataset. PassGAN was trained on the Russian email -based dataset and was found that PassGAN did not perform very well when trained on the dataset. After providing suitable explanations for the above mentioned result, the paper concludes by stating some key points to be remembered while training GANs on any type of data. The paper first introduces various kinds of data breaches using attacks like dictionary attack, brute force, rainbow table attack, etc. Dictionary attacks can be described as guessing passwords by using a specialized wordlist .

Brute force attack uses scripts to generate passwords systematically. But, the downside is that for every increasing character, time taken by brute force to crack password, increases exponentially. Rainbow table attacks store pre-computed hash values so that time taken to crack a password significantly reduces. Next, the paper explores various causes for data breaches. The most common is weak credentials or choosing the same passwords for multiple websites. In phishing, the attacker tries to attack by disguising as a legitimate entity to gain access to information. Social engineering involves using manipulating tricks to get sensitive information. Ransomware is becoming increasingly popular where data of an organization is infected, and restricted until a ransom amount is paid. After the model is trained on the Russian email-based dataset, few interesting results were obtained. The most interesting result was the fact that PassGAN was able to detect passwords based on reverse translation technique, where russian words were written in english.

These words may seem gibberish to english-speaking users but they are easily recognized by russians. Analysing the results, the paper concluded that PassGAN did not perform very well when trained on this particular dataset.

III. Proposed System

This paper aims to use generative, unsupervised model like GAN to generate more likely password guesses. The GAN model consists of two networks called Generator and Discriminator.
Generator is a network that is trained to generate passwords by learning the distribution of the input dataset. Discriminator labels the passwords generated by generator as real passwords or generated passwords. Model is said to be trained when the discriminator is unable to label the password as real password or generated password. After training, discriminator is removed and generator is used to generate a password dataset that can be used to crack passwords. Also, discriminator can be used to evaluate the strength of a given input password.

The training dataset quality must be good and the volume of the dataset must be high. Also, fast cpu/gpu must be available to train the model for many epochs.The model uses some of the tensorflow and keras inbuilt libraries for implementation.

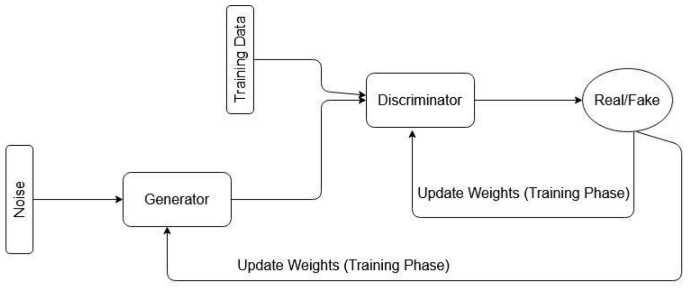
Training Data
Discriminator
Real/Fake
Noise
Generator
Update Weights (Training Phase)
Update Weights (Training Phase)

Fig. 2. Technical Flowchart of the System

The performance of the model is constrained by the amount and quality of the dataset being used to train the model.

IV. Implementation

The createDictionary method takes as input the dict.txt file that has mapping of every character to it's respective integer value. First, the file is opened, read and split by newline as the delimiter. The, every line in the list is split by considering space as the delimiter. Now, the first value gives the character and the second value is the corresponding integer mapping. Later, two dictionaries are created – one for encoding and other for decoding. The input passwords must be encoded to a list of integers for the model to train. To achieve this, a function Encoding String is defined that takes a password in string format as the input. First, the length of the password is computed. If the password is less than the maximum length allowed, then the remaining values in the output list must be padded with a character. After the passwords are generated from the model, the generated list of integers must be decoded into passwords in human-readable format. To achieve this, decodeList function is defined that takes the encoded list as the input. A variable decodedString is declared and initialized with a NULL value.

Generator takes a low dimension noise vector as an input and outputs a higher dimension vector. This output is then decoded to a string(password). Generator has four dense layers and has tanh as the activation function in the final layer. While training the generator, the number of epochs is taken as 1000. Then, the TensorFlow modules are imported, followed by importing Adam optimizer module.
From layers module of keras, Sequential, Dense, Relu modules are imported. Then, Input module, Mode modules are imported. First, the training data is normalized and mapped to a range from -1 to 1 using a range mapping formula. Then, from the training data train size and input size is extracted. Then, noise dimensions and batch size parameters are defined. steps per epoch is computed by dividing training size by batch size. Adam optimizer is initialized with learning rate as 0.02.

Discriminator takes an encoded vector as an input and outputs whether the input is real or fake (generated by generator). It takes a high dimension vector as input and outputs a single value. Discriminator has 4 dense layers and has sigmoid as the activation function in the final layer. Discriminator has a keras sequential layer as the input layer. Then, it has a sequential fully connected dense layer with 40 neurons. Followed by that, there is a Dense layer of 20 neurons. And the output layer is a dense layer with a single neuron that uses the sigmoid activation function. The optimizer used is Adam and the loss function is binary cross entropy.

The loss function is chosen according to the activation function. Since sigmoid is a classification function, binary cross entropy as loss function would perform the best. Also, the output layer has a classification activation function since the output generated by the discriminator has to be classified between real or generated. If the output is a value between 0 and 0.5, it is considered as a real password. If it is between
to 1, it is a generated password.

In GAN Module, the generator and discriminator functions are invoked and are combined to form a GAN model. Then, the model is trained by specifying the number of epochs and batches. In the training module, actual training of the generator and discriminator modules take place. For every epoch, steps per epoch is computed which forms the number of batches. And for every batch, batch size number of instances are made as the noise vector. The noise vector is obtained by generating random normal samples between 0 and 1. The noise vector is fed as input to the predict function of generator which outputs the list of samples generated. Next, batch size number of instances are picked from the training dataset and these samples form the real dataset. Both real and generated datasets are combined using concatenate method of numpy and the combined vector is fed as input to the discriminator.

To generate the label vector, first a zero vector of size twice of batch size is created. Then, the last batch size number of instances are made as one that results in a vector with instances corresponding to real data labelled as one and instances corresponding to generated data are labelled as zero. Next, train on batch function is used to train the discriminator, which has two parameters - the combined vector and the combined label vector. This training function outputs a loss after training that forms the discriminator loss. The labels for the generator must be a vector of all ones that indicates to the generator that the random noise vector gives as the input is the real data for the generator. So, a numpy vector of batch size is created and the numpy vector as well as the noise vector is fed as the input to train on batch function for the GAN model. Since the discriminator had been set to untrainable, this trains only the generator. After every epoch, the generator and discriminator losses are printed to give an idea of the accuracy of the model after every epoch.

In the final part of the implementation, the model is tested by generating passwords using the generator and if the password generated matches with the distribution of the input dataset, then the model was successful in training and learning the distribution of the input dataset. First, a function round off is defined that takes as input the integer encodings and rounds them off to the nearest value. This is needed because passwords generated could possibly have decimal places and to decode the, back to their character encodings, rounding off is necessary.

Further, a map range function is defined that maps the passwords to a specific range. This range is same as the range chosen while creating the mapping dictionary. The map range is required because the passwords generated by the generator are not limited to a specific range. The generate passwords function takes as input the random noise vector and prints the generated passwords. First, the predict method of generator model is used to generate passwords. Then, the generated passwords are mapped from a range of -1,1 to the initially chosen dictionary range. For every password in this list of generated passwords, deocodeList method is applied to convert list of integer representations to the corresponding passwords in the text format.

The trained model must be saved so that we can load the model in the future without training it again. To achieve this, the models are first converted to json file using to json method. Later, a new json file is created and the json is written to this file. Finally, the model weights are saved in a .h5 file format using save weights method. Before loading the model, the optimizer has to be initialized and mode from json has to be imported from model module of keras. The saved json file is opened in read mode and all the contents are read into a variable. The, the file is closed. The model is constructed from the json using model from json method. Then, the weights are loaded from the .h5 file using the load weights method. Finally, the model is compiled by specifying the loss and the optimizer as the parameters to the compile method. This completes the loading process and now the model can be directly used without the need for re-training it.

The generated model is verified by checking if the passwords generated by the generator matches the input dataset. If it does, then the generator has successfully learnt the patterns in the distribution of the dataset. And, when trained with some other dataset with a different distribution, the generator is likely to learn the distribution of the new dataset as well.

Discriminator is tested based on the ability to discriminate between real and fake passwords. To test the discriminator, a list of passwords with equal number of real and generated passwords are fed as input to the discriminator and the predict method is used to get the discriminator predictions. If the discriminator outputs a value between 0 and 0.5 for fake passwords and values between 0.5 and 1 for real passwords then the discriminator is said to be trained as it's able to distinguish between real and fake passwords by observing the difference in their patterns.

After testing the generator and discriminator models individually, both the models are combined and are subjected to testing. In this process, first a train list is created and is populated with real passwords. Similarly, a generated list is created and is populated with generated passwords. The train list is iterated and for every password in the list, the password is encoded and is saved to a new list. Similarly, for generated list, every password is encoded and is saved to a new list. Now, both these lists are combined to a list and it's fed as input to the predict function of discriminator. For the model to be trained, all the numbers in the output of the predict function must be similar. That is, of the outputs are indistinguishable, then the discriminator has failed to separate the real passwords from the fake generated passwords. This indicates that the generator is successful in generating real-like passwords that fooled the discriminator. Hence, the combined model is said to be trained and can be used to generated new passwords that match the distribution of the dataset.

V. Experimental Results

This section discusses in detail the implementation of the model and the results obtained.

For training the model, rockyou dataset is used. RockYou contains about 32 million passwords obtained from popular data breaches. Considering the training time and processing power required, only topmost 10k passwords from RockYou was used for training. Since ML models cannot work with characters, every password in the input dataset had to be encoded into a list of integers before being fed to the model. To achieve this, a dictionary of all ascii characters was created that mapped a character to a corresponding integer. Using this dictionary, all passwords from input were encoded to lists and were stored in a numpy array. This completes the preprocessing part of the implementation.

Generator was implemented using multiple Dense Sequential layers with Adam optimizer and tanh activation function used at the output layer. Discriminator was designed similarly using multiple Dense layers with Adam Optimizer. And the output layer had one neuron for spitting out a probability of password being real or fake. Since this becomes a classification problem, sigmoid activation function was used at the output layer with binary crossentropy as the loss function. Both the models were combined into a single model and discriminator was made untrainable so that when the model back propogates the gradient, only generator is trained. While training the model, at every epoch, the input dataset is split into multiple batches and in every batch, fake samples are the output of passwords generated by the generator and real samples form the passwords picked randomly from the input dataset. All the fake samples are labelled as zero and all real samples are labelled as one and are fed to discriminator. Then, discriminator is trained using the combination of passwords and labels and finally the gan model is trained using a random noise vector, that trains the generator. Discriminator is excluded from training because it was made untrainable at the beginning of the training process. This configuration was trained for 100 epochs with batch size as 16. It was found that both generator and discriminator losses started at 1.1 and later reduced to around 0.2. Also, it was found that when after every epoch, when generator loss decreases, the discriminator loss increases as expected because both generator and discriminator form an adversarial system.

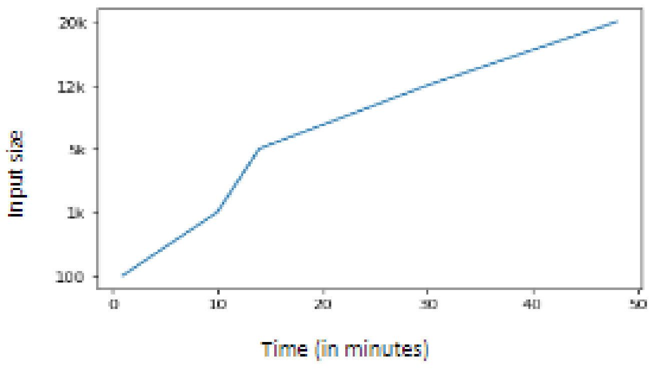

Fig. 3. Input size vs training time

For testing the model, a random noise vector is given as input to the generator and it generates a list of passwords. For every password in the list, the generated integer encoding are first rounded off followed by a mapping to a specific range. Later, all the passwords are decoded from their integer representation to a sequence of characters. And this final list of passwords forms the passwords generated by the model after learning the input distribution. As mentioned above, discriminator is used to discriminate between real and generated passwords. To test if discriminator has learnt to do this, equal number of real and generated passwords were combined into a list without any labels. The list was fed as input to the discriminator so that it gives out a single value for every password. It was found that among generated and real passwords, the value given out by discriminator was nearly the same and all values were around 0.4. This indicates the discriminator is not able to distinguish between real and fake passwords and generator was successful in generating real like passwords. Therefore, it has been verified that both discriminator and generator are trained completely and can now be used with different input data sets to learn their password distribution.

To verify if the model works as expected, the trained model was further subjected to unit testing and performance testing. In unit testing, all the methods were tested using unittest module in python. This helped in the correction of a bug in the encoding function. The bug was that the actual password length with which the model trains was one more than the expected password length. For performance testing, the training time for various datasets were compared using a plot. Fig.4 shows the time taken against the input size. Clearly, as the input password size increases, the time taken had increased linearly. The information from this plot when combined with the GPU speed gives us a clear picture of the various sizes of input that can be trained using the model. The above discussed testing methodologies have made the model more robust and bug free.

VI. Conclusion and Future Works

In this paper, a password generative technique based on GAN was implemented, trained and verified using various testing methodologies. The main advantage of this model when compared to other rule based techniques is that it doesnot rely on any additional information or assumptions on the input dataset. The performance of the model was evaluated using a two step process. Generator was evaluated by allowing it to generate a set of passwords and these were matched against the passwords from the input data set and was found that the generated passwords matched the distribution of the input passwords. Discriminator was evaluated by passing a combined list of generated and real passwords as the input and it was found that discriminator failed to distinguish generated passwords from the real ones. The only disadvantage is that when compared with the other methods, the implemented model requires to output a large number of passwords to start generating meaningful passwords. But, this cost would be negligible when trained on a large dataset.

In the future, the implemented model can be combined with the existing methods like HashCat to further increase the number of passwords generated. First, the input dataset can be passed to HashCat that would crack most of the easy passwords and the passwords which could not be cracked by HashCat can be fed as input to the GAN model. Also, other alternative models like RNNs or Autoencoders could be implemented and compared with GAN. Finally, the existing model could be trained with other kinds of datasets like linkedin leaked or Ashley Madison datasets and verify if the model is able to learn the distribution when trained with these data sets.

References

[1] S. Nepal, I. Kontomah, I. Oguntola, and D. Wang, "Adversarial password cracking,"

[2] A. Chen, "Presenting new dangers: A deep learning approach to password cracking,"

[3] B. Hitaj, P. Gasti, G. Ateniese, and F. Perez-Cruz, "Passgan: A deep learning approach for password guessing," in *International Conference on Applied Cryptography and Network Security*, pp. 217–237, Springer, 2019.

[4] V. Garg and L. Ahuja, "Password guessing using deep learning," in *2019 2nd International Conference on Power Energy, Environment and Intelligent Control (PEEIC)*, pp. 38–40, IEEE, 2019.

[5] S. Nam, S. Jeon, and J. Moon, "Generating optimized guessing candidates toward better password cracking from multi-dictionaries using relativistic gan," *Applied Sciences*, vol. 10, no. 20, p. 7306, 2020.

[6] A. Malik, "Survey paper on applications of generative adversarial networks in the field of social media," *International Journal of Computer Applications*, vol. 975, p. 8887.

[7] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[8] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, 2017.

[9] S. Nam, S. Jeon, and J. Moon, "A new password cracking model with generative adversarial networks," in *International Workshop on Information Security Applications*, pp. 247–258, Springer, 2019.

[10] Y. Liu, Z. Xia, P. Yi, Y. Yao, T. Xie, W. Wang, and T. Zhu, "Genpass: A general deep learning model for password guessing with pcfg rules and adversarial generation," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2018.

[11] R. Hranick ỳ, L. Zobal, O. Ryšavỳ, and D. Kolář, "Distributed password cracking with boinc and hashcat," *Digital Investigation*, vol. 30, pp. 161–172, 2019.

[12] S. Nam, S. Jeon, H. Kim, and J. Moon, "Recurrent gans password cracker for iot password security enhancement," *Sensors*, vol. 20, no. 11, p. 3106, 2020.

[13] D. Biesner, K. Cvejoski, B. Georgiev, R. Sifa, and E. Krupicka, "Generative deep learning techniques for password generation," *arXiv preprint arXiv:2012.05685*, 2020.

[14] I. K. Dutta, B. Ghosh, A. Carlson, M. Totaro, and M. Bayoumi, "Generative adversarial networks in security: A survey," in *2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pp. 0399–0405, IEEE, 2020.