Semi-supervised Conditional GAN for Simultaneous Generation and Detection of Phishing URLs: A Game-theoretic Perspective

Sharif Amit Kamran*, Shamik Sengupta† and Alireza Tavakkoli‡
University of Nevada, Reno NV, USA
skamran@nevada.unr.edu*, ssengupta@unr.edu†, tavakkol@unr.edu‡

Abstract—Spear Phishing is a type of cyber-attack where the attacker sends hyperlinks through email on well-researched targets. The objective is to obtain sensitive information by imitating oneself as a trustworthy website. In recent times, deep learning has become the standard for defending against such attacks. However, these architectures were designed with only defense in mind. Moreover, the attacker’s perspective and motivation are absent while creating such models. To address this, we need a game-theoretic approach to understand the perspective of the attacker (Hacker) and the defender (Phishing URL detector). We propose a Conditional Generative Adversarial Network with novel training strategy for real-time phishing URL detection. Additionally, we train our architecture in a semi-supervised manner to distinguish between adversarial and real examples, along with detecting malicious and benign URLs. We also design two games between the attacker and defender in training and deployment settings by utilizing the game-theoretic perspective. Our experiments confirm that the proposed architecture surpasses recent state-of-the-art architectures for phishing URL detection.

Index Terms—Generative Adversarial Networks; Phishing Attacks; Phishing URL detection; Cyber Security; Phishing

I. INTRODUCTION

Historically, cyber-security companies have used machine learning as a defensive measure to identify malicious websites and suspicious network activity. Given the massive surge in phishing attacks in recent times, it propelled companies to advertise novel systems for detecting inbound threats. According to the FBI [1], phishing was the most common type of cybercrime in 2020 and phishing incidents nearly doubled from 114,702 incidents in 2019 to 241,324 incidents in 2020. Out of this, 65% of active hacker groups relied on spear-phishing to carry out the primary infection vector as reported in [2].

In another report by Verizon [3], the attackers used email as a carrier for 96% of the targeted attacks. While sites such as Phishtank [4] exist for sharing and verifying plausible malicious or phishing URLs, there has been little exploration into how attackers can also utilize machine learning to their advantage for generating such URLs.

The necessity for efficient countermeasures has made spear phishing a popular field of research in recent times. Effectively, two main types of methods for phishing URL detection have emerged: (i) Database of Blacklists and Whitelists URLs [5], (ii) Character or feature level URL detection [6], [7]. The database approach is quite inefficient due to not being updated on time or overlooking possible attacks from new URLs. The second approach is quite convenient for possible new attacks due to extracting latent representation of the feature level information and utilizing that to train a deep learning models (RNN [8], LSTM [9], CNN [10], GAN [11]) to make the prediction. Compared to character-level detectors, the feature-level detection system suffers from out-of-distribution predictions because the features are constrained. As a result, the current state-of-the-art approaches utilize deep learning to learn from distinct characters of URLs to improve detection performance and enable zero-day phishing defense [6], [8], [12].

Cyber-security companies design this system with only defense in mind. The underlying motivation and perspective of the attacker are not taken into account, so the system can’t formulate the best defense strategy for current or future attacks. We need a game-theoretic approach to understand the rational decision-making process of the attacker and how the defender should counter-play in this game. By taking all these into we propose the following in this paper:

- We propose a Conditional Generative Adversarial Network with novel training strategy for simultaneous generation and detection of phishing URL, trained in a semi-supervised manner.
- We design two novel games based on game theory, i) between the pseudo attacker and the discriminator for training our conditional GAN, and ii) between the real hacker and the defender (the discriminator) for deployment in the wild.
- Our model outperforms other state-of-the-art architectures by learning from real and adversarial phishing URLs.
- Our discriminator detects URLs in real-time and is robust against adversarial examples.

II. LITERATURE REVIEW

Over the years, cyber-security firms have extensively utilized Machine Learning for detecting Phishing URLs and malicious sites. These works ranged from SVM [13], [14], [15], Random Forest [16], [17], KNN [18], [19] and a combinations of this techniques[20], [21]. After the advent of Deep Learning, Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) have become the norm for automated feature extraction from a huge volume of data without any human intervention. Some recent works involving deep learning are based on CNN [22], [23], [10], CNN with Multi-headed Attention.
Fig. 1: Proposed Conditional GAN consists of a single Generator and Discriminator where the Generator takes the Real Benign or Malicious URL, a noise vector, and the class labels as input. The Generator outputs a synthesized URL, whereas the Discriminator categorizes Benign or Malicious URL, and detects adversarial examples.

With the popularity of Generative adversarial networks (GAN), there has been a surge in NLP applications ranging from text generation [27], [28], [29], language models, [30], [31] and text classification [32], [33]. The reason being, GANs can extract and learn fine-grained information from texts and utilize that to create synthetic examples using a generator architecture. Along with that, it utilizes a discriminator architecture for distinguishing between real and synthetic examples. By leveraging the generator and discriminator, the GAN can be trained in an adversarial setting with multiple cost functions and weights. Generative networks has also been used for synthesizing and identifying malicious URLs from lexical [12], [34], [35] or character-level data [36], [6]. The latter work involves training the generator to synthesize each URL character by sampling from a random noise vector, and the architecture is called unconditional GAN. This type of GAN suffers from "mode collapse" [37], [38] due to sampling from only a noise vector. As a result, the generator creates garbage data that the discriminator fails to distinguish, leading to nonconvergent adversarial loss. In contrast, conditional GAN incorporates both original data (malicious or benign URL) and label with the noise vector as input. As a result, it improves the overall model training and helps to synthesize more realistic output.

Another major problem with the current state-of-the-art architecture is that they heavily emphasize defensive measures, somewhat ignoring the attacker’s perspective in this scenario. A cyber-defense scenario almost always depends on game theory to understand the attacker’s motives and perspective for maximizing the defender’s reward. Many similar games have been designed for phishing URL detection with humans in the loop [39], [40], [41] and adversarial games for generative networks [42], [43], [44]. However, no work has been proposed, combining adversarial components of GAN and game-theoretic perspectives of attacker-defender for detecting phishing URLs. To alleviate this, we propose a novel conditional GAN that detects phishing URLs and is robust against adversarial examples. Furthermore, we design two games, one for training our architecture and the other for deploying it to a real environment for better understanding the attacker’s perspective. Our quantitative result proves that the proposed technique surpasses state-of-the-architectures and synthesizes realistic benign and malicious URLs.

III. THE PROPOSED METHODOLOGY

We propose a conditional generative adversarial network (GAN) comprising a generator for producing adversarial phishing URLs and a discriminator for robust classification of malicious/benign URLs. We also introduce two non-cooperative and non-zero-sum games between the i) Pseudo Attacker & Discriminator and ii) Attacker & the Phishing URL Detector. By formalizing the game concept and addressing the limitations of current systems, we illustrate how generative networks can produce robust results in Phishing URLs detection and synthesis tasks. First, we elaborate on the proposed architecture in Sections III-A, III-B, III-C. Next, we structure the two games
based on players, their utilities, and action spaces in Section III-D III-E.

A. Generator

Combining generative networks with an auxiliary classification module has been shown to produce realistic text-generation and accurate class prediction as observed in [32, 45]. We adopt a similar architecture to Auxiliary Classification GAN [45] by incorporating a class conditioned generator and a multi-headed discriminator for categorical classification and adversarial example recognition as illustrated in Fig. 1. The generator concatenates the original URL, x, label, y, and a noise vector, ζ as input and generates \( G(x, \zeta, y) \) malicious/benign URL. The noise vector, z, is smoothed with a Gaussian filter with \( \sigma = 3 \) before using it as input. The generators consist of Convolution, Transposed Convolution, Batch-Normalization, and Leaky-ReLU activation layers. We use convolution for downsampling \( 3 \times \) and use transposed convolution to upscale \( 3 \times \) again to make the output resolution the same as the input. For regular convolution we use kernel size, \( k = 3 \), stride, \( s = 1 \) and padding, \( p = 1 \). For downsampling convolution and transposed convolution we use kernel size, \( k = 3 \), stride, \( s = 2 \) and padding, \( p = 1 \). Convolution and Transposed Convolutions are followed by Batch-normalization and Leaky-ReLU layers. The encoder consists of six convolution layers. The number of features are \([F1, F2, F3, F4, F5, F6]= [32, 32, 64, 64, 128, 128]\). The decoder consists of three transposed convolution layers, and they have \([D1, D2, D3]= [128, 64, 32]\) number of features. The generator has an input and output dimension of \( 200 \times 67 \) and incorporates Sigmoid activation as output.

B. Discriminator

In contrast, the discriminator takes both real URL x and adversarial URL \( G(x, \zeta, y) \) as input sequentially. It simultaneously predicts if the example is real or adversarial and classifies the signal as either Benign or Malicious. The discriminator consists of Convolution, Batch-Normalization, Leaky-ReLU activation, and Dense layers. We use convolution for downsampling 3 times. For convolution we use kernel size, \( k = 3 \), stride, \( s = 1 \) and padding, \( p = 1 \), except for downsampling convolution where we use stride, \( s = 2 \). The convolution layer is succeeded by Batch-normalization and Leaky-ReLU layers. After that, we use two fully connected layers. The encoder consists of six convolution layers and two dense layers. The number of features are for each layer is \([F1, F2, F3, F4, F5, F6]= [16, 16, 32, 32, 128, 128, 256, 64]\). We use two output activation: classification with Softmax activation for Benign/Malicious classes and Sigmoid activation for real/adversarial example detection.

C. Weighted Cost Function and Adversarial Loss

We use LSGAN [47] for calculating the adversarial loss and training our Generative Network. The objective function for our conditional GAN is given in Eq. 1.

\[
\mathcal{L}_{adv}(G, D) = \mathbb{E}_{x,y} [(D(x,y) - 1)^2 + \mathbb{E}_{x,y} [(D(G(x,\zeta,y),y) + 1)^2]]
\]

In Eq. 1 we first train the discriminators on the real URLs, x. After that, we train with the synthesized URLs, \( G(x,\zeta,y) \). We begin by batch-wise training the discriminators \( D \) on the training data for couple of iteration. Following that, we train the G while keeping the weights of the discriminators frozen. For classification of Benign and Malicious URLs, we use categorical cross-entropy as in Eq. 2.

\[
\mathcal{L}_{class}(D) = -\sum_{i=0}^{k} y_i \log(y'_i)
\]

The generators also incorporate the reconstruction loss (Mean Squared Error) as shown in Eq. 3. By incorporating the loss, we ensure the synthesized URLs as convincing as the real URLs.

\[
\mathcal{L}_{rec}(G) = \mathbb{E}_{x,y} \|G(x,\zeta,y) - x\|^2
\]

By incorporating Eq. 1, 2 we can formulate our final objective function as given in Eq. 4.

\[
\min_{G,D} \left( \max_{\lambda_{adv}} \mathcal{L}_{adv}(D) + \lambda_{rec} \mathcal{L}_{rec}(G) + \lambda_{class} \mathcal{L}_{class}(D) \right)
\]

Here, \( \lambda_{adv}, \lambda_{rec}, \) and \( \lambda_{class} \) implies different weights, that is multiplied with their respective losses. The loss weighting decides which architecture to prioritize while training. For our system, more weight is given to the \( L_{class}(D) \) for better categorical prediction, and thus we select bigger \( \lambda \) value for this.

D. Game during Model Training

We begin by contextualizing a novel two-person game similar to [48], using the proposed conditional GAN between the Pseudo Attacker and the Phishing URL detector. Both the players are interlocked in a non-cooperative, non-zero-sum game signified formally by the following tuple of players, action spaces, and utility variables: \( \text{Game}_{MT} = \{P, D\}, \{\Theta_G, \Theta_D\}, \{u_G, u_D\} \) where

- \( P \) is the pseudo-attacker who controls generator G by utilizing its \( \theta_G \)-parametrized function approximator, \( G(x,\zeta,y) \) with \( G: \mathbb{R} \rightarrow \mathbb{R}^d \), to choose its actions,
- \( D \) is the phishing URL detector, which uses a \( \theta_D \)-parametrized function approximator, \( D(G(x),y) \) with \( D: \mathbb{R}^d \rightarrow \mathbb{R} \), to choose its actions,
- \( \theta_G \) is the action space of the generator, where \( \theta_G \in \Theta_G \),
- \( \theta_D \) is the action space of the discriminator, where \( \theta_D \in \Theta_D \),
- \( u_G \) and \( u_D \) are the generator and discriminator's utility variables.

Based on the assumptions above, we can rewrite the Eq. 1 as Eq. 5. Here \( p(x) \) is the distribution of the given data samples, \( x \in \mathbb{R} \), \( p(\zeta) \) is the Gaussian distribution of the noise sample, \( \zeta \in [0,1] \) and \( p(y) \) is the distribution of the ground truth, \( y \in \{0,1\} \).

\[
u_{adv}(\theta_G, \theta_D) = \mathbb{E}_{x \sim p(x), y \sim p(y)} [(D(x,y) - 1)^2 + \mathbb{E}_{x \sim p(x), \zeta \sim p(\zeta), y \sim p(y)} [(D(G(x,\zeta,y),y) + 1)^2]
\]

\[
u_{rec}(G) = \mathbb{E}_{x,y} \|G(x,\zeta,y) - x\|^2
\]

\[
u_{class}(D) = -\sum_{i=0}^{k} y_i \log(y'_i)
\]
The phising URL detector can also distinguishes between "Benign" and "Malicious" URL using categorical-cross entropy loss in Eq. 2. Moreover, the generator also tries to fool the discriminator with realistic sample using L2 bounded reconstruction loss in Eq. 3. We can rewrite Eq. 2 and Eq. 3 and formulate it into, Eq. 6 and Eq. 7. By combining Eq. 5, Eq. 6, and Eq. 7 we can find the final utility function given in Eq. 8. Here, $\lambda$ is the weight values of each utility functions similar to weight values in Eq. 4.

$$u_{\text{class}}(\theta_G, \theta_D) = \mathbb{E}_{y \sim p(y), y' \sim p(y')} \left[ - \sum_{i=0}^{k} y_i \log(p'_i) \right]$$  \hspace{1cm} (6)

$$u_{\text{rec}}(\theta_G, \theta_D) = \mathbb{E}_{x \sim p(x), \zeta \sim p(\zeta), y \sim p(y)} \left[ \|G(x, \zeta, y) - x\|^2 \right]$$  \hspace{1cm} (7)

$$u(\theta_G, \theta_D) = \lambda_{\text{adv}} [u_{\text{adv}}(\theta_G, \theta_D)] + \lambda_{\text{rec}} [u_{\text{rec}}(\theta_G, \theta_D)] + \lambda_{\text{class}} [u_{\text{class}}(\theta_G, \theta_D)]$$  \hspace{1cm} (8)

Next, we design a Kuhn tree between the Pseudo Attacker (Attacker) and the Phishing URL detector (Defender) as illustrating in Fig. 2. We assume the strategy space of the Pseudo Attacker consists of two actions: providing (i) adversarial example $G(x, \zeta, y)$ and (ii) real example $x$. In contrast, the Phishing URL detector can take four actions: classify the sample as (i) Benign Fake, (ii) Malicious Fake, (iii) Benign Real, or (iv) Malicious Real. We further assume that the Pseudo Attacker will make the first move, and then the Phishing URL detector will follow by taking one of the four possible actions. It should also be noted that the Phishing URL detector does not have information about the sample’s type and if they are real or fake.

The game has eight possible outcomes, based on the actions taken by the Attacker and Defender. Suppose the attacker uses the adversarial example $G(x, \zeta, y)$, then the defender can use any of the four actions to classify it. We can see in Fig. 2 the best strategy for the defender would be to classify it as either Malicious Fake or Benign Fake. On the other hand, if the attacker uses the real example $x$, then the defender’s best strategy would be to classify it as either Malicious Real or Benign Real. The payoff for the attacker and the defender were created using the $\lambda$ values in Eq. 8. The maximum profit the attacker can gain by fooling the Phishing URL detector (discriminator) can be $\lambda_{\text{rec}} = 10$. The minimum can be 0.

Contrarily, the defender would get a maximum value of 11, by summing $\lambda_{\text{cls}} = 10$ and $\lambda_{\text{adv}} = 1$. The minimum payoff would be 10, because of the correct class prediction.

**E. Game after Model Deployment**

Next, we design a novel two-person game for model deployment, which would be played between the real-life attacker and the Phishing URL detector (Defender) in a non-cooperative, non-zero-sum game signified formally by the following tuple of players, action spaces, and utility variables: $Game_{MD} = \{A, D\}, \{\Theta_A, \Theta_D\}, \{u_A, u_D\}$ where

- A is the attacker who either sends and doesn’t send the phishing URL and the action is defined by $\theta_A \in \{\text{Dont Send, Send}\}$.
- D is the phishing URL detector, which either classifies the URL as Benign or Malicious and the actions is defined by $\theta_D \in \{\text{Benign, Malicious}\}$.
- $\theta_A$ is the action space of the attacker, where $\theta_A \in \Theta_A$.
- $\theta_D$ is the action space of the discriminator, where $\theta_D \in \Theta_D$.
- $u_A$ and $u_D$ are the attacker and phishing URL detector’s utility variables.

Based on this above assumption we create our Kuhn tree for Model Deployment and is illustrated in Fig. 3. This time the strategy space of the attacker consists of two actions: either to (i) Send Phishing URL or (ii) Don’t Send. Contrarily, the
Phishing URL detector can take two actions: classify the sample as (i) Benign or (ii) Malicious. The assumption also remains the same: the attacker will make the first move, and then the Phishing URL detector will follow by taking one of the two possible actions. The utility values are such that \( u_A \in \mathbb{R} \) and \( u_D \in \mathbb{R} \).

The game has three possible outcomes, based on the actions taken by the attacker and defender. If the attacker does not send the phishing URL, they will both get a payoff of 0. On the other hand, if the attacker sends the phishing URL, the defender either has to classify it as malicious or benign. We can see in Fig. 5 the best strategy for the defender would be to classify it as Malicious. The payoff for the attacker and the defender were created using arbitrary values. The minimum payoff would be -3, because it would mean losing all the sensitive information.

IV. EXPERIMENTS

In the next section, we detail our model experiments and evaluate our architecture based on quantitative metrics. First, we elaborate on our detailed algorithm in Sec. [V-A]. Then detail our hyper-parameter selection and tuning in Sec. [V-B]. Next, we describe our adversarial training scheme in Sec. [V-C]. Also, we compare our architecture with existing state-of-the-art models based on some quantitative evaluation metrics in Sec. [V-D].

A. Dataset

We used the publicly curated and standardized dataset provided in Kaggle [49]. It consists of 500,000+ benign and malicious URLs collected from various sources. Next, we sample 50,000 benign and malicious URLs with a max length of 200 characters. The dataset includes an equal portion of benign and malicious websites. We use a dictionary of 67 unique characters (twenty-six alphabets, ten digits, and thirty-one special characters) to convert the data into a one-hot encoding matrix. Now, each URL ends up being a 2-D matrix of one special characters) to convert the data into a one-hot encoding matrix.

B. Hyper-parameter tuning

For adversarial training, we used LS-GAN loss [47]. We picked \( \lambda_{rec} = 10 \) and \( \lambda_{class} = 10 \) (Eq. 4). For optimizer, we used Adam [50], with learning rate \( \alpha = 0.0002 \), \( \beta_1 = 0.5 \) and \( \beta_2 = 0.999 \). We train with mini-batches with batch size, \( b = 64 \) for 200 epochs. It took approximately 6 hours to train our model on NVIDIA GPU. Code Repository Link.

C. Training procedure

In this section, we elaborate on our detailed algorithm provided in Algorithm 1. To train our model, we start by initializing all the hyper-parameters. Next, we sample a batch of the real URLs \( x \). We train the real URLs \( D \). After that, we use \( G \) to synthesize fake URLs \( G(x, \zeta, y) \) and use them for training our discriminator \( D \) again. We train the discriminator in this manner for a couple of iterations. Following that, we calculate the adversarial loss, \( L_{adv} \), the classifier loss, \( L_{class} \), and update the weights. We freeze the weights of the discriminator. Next, we train the generator and calculate the \( L_{rec}(G) \) and update the generator’s weights. In the final stage, while keeping the discriminator’s weights frozen, we jointly fine-tune the discriminator and generator together. We calculate the total loss by adding and multiplying with their relative weights. For testing, we save the snapshot of the model and its weights.

D. Quantitative Evaluation

For finding the character similarity with the original URL, we benchmark synthesized adversarial URLs using four different metrics, i) Mean Squared Error (MSE), ii) Structural Similarity Index (SSIM), and iii) Normalized Mean Squared Error (NRMSE). Table. I shows that SSIM for the test set has a 98.33% score, which means the adversarial examples are structurally similar to the original signal. For MSE and NRMSE, our model generates quite similar adversarial examples to real ones. It is important to note that we want to achieve lower MSE and RMSE. Similarly, we want to score higher for SSIM.

For benign and malicious URL classification tasks, we compare our model with other state-of-the-art architectures as given in Table. II. We further experiment on adversarial examples synthesized using our Generator from the above tests set, which we show in Table. III. We use Accuracy
Fig. 4: Our model outperforms other methods in terms of Area under the Receiver Operating Curve (ROC) for detecting Benign/Malicious Phishing URLs.

(ACC), Sensitivity, Precision, F1-Score, and AUC (area-under-the-curve) for metrics. We can see for the first experiment, except for the architectures given in [24], our model achieves the best score compared to other deep learning and machine learning derived architectures. The architecture in [24] uses 1D CNN and LSTM architectures. Because these architectures have millions of parameters, they might be overfitting on the data. Compared to them, our architecture is quite lightweight and has around 81,000 parameters. Our model also has an inference speed of 0.64 ms (milisecond). Meaning it can process around 1500+ URLs in 1 second. The models in [14], [6] uses DCGAN with character level data. However, they perform poorly compared to our model. We also provide AUC and illustrate all the model’s Receiver operating curve (ROC) in Fig. 4.

For our last experimentation, we evaluate our discriminator and check its accuracy to detect adversarial examples as given in Table. III. We use a test set with 50% adversarial and 50% real data to carry out this benchmark. By combining the real and the adversarial examples synthesized by our generator, we can have this 50/50 test split. So from the test set of 10,000 URLs, we end up having 20,000 samples. We use Accuracy, Sensitivity, Specificity, F1-score, and AUC as standard metrics for measuring our model’s performance. As shown in Table. III our model is quite robust against adversarial examples. However, the model performs better for detecting real examples compared to adversarial examples, which is validated by 100% Specificity. Moreover, the model detects 74.9% adversarial examples correctly, which is validated by the

| Method             | ACC    | Sensitivity | Specificity | Precision | F1-score | AUC    |
|--------------------|--------|-------------|-------------|-----------|----------|--------|
| Proposed Method    | 0.9552 | 0.9600      | 0.9508      | 0.9554    | 0.9552   |
| SVM [12]           | 0.9135 | 0.9210      | 0.9073      | 0.9141    | 0.9135   |
| Random Forest [23] | 0.8682 | 0.8508      | 0.8525      | 0.8664    | 0.8638   |
| CNN [24]           | 0.9251 | 0.9190      | 0.9303      | 0.9246    | 0.9251   |
| LSTM [24]          | 0.9471 | 0.9522      | 0.9425      | 0.9473    | 0.9471   |

TABLE II: Phishing URL classification : Comparison of architectures trained and tested on real Phishing URLs

V. CONCLUSION

In this paper, we introduced a conditional Generative Adversarial Network with novel game-theoretic training strategy for simultaneous synthesis of adversarial examples and detecting Phishing URLs. Our architecture outperforms previous techniques by adopting a semi-supervised learning scheme of synthesizing adversarial URLs while predicting their category. The model is best suited for real-time Phishing detection monitoring both locally (client-end) and globally (server-end), where it can perform robustly and effectively. One future work is to test the adversarial examples on other architectures and compare their robustness.

REFERENCES

[1] “Internet crime report,” May 2020. [Online]. Available: $www.ic3.gov/$
[2] “Internet security threat report,” February 2019. [Online]. Available: $docs.broadcom.com/doc/istr-24-2019-en/$
[3] “Data breach investigations report,” 2020. [Online]. Available: $enterprise.verizon.com/en-gb/resources/reports/dbir/$
[4] “Phishtank.” [Online]. Available: $phishtank.org/$
[5] A. K. Jain and B. B. Gupta, “A novel approach to protect against phishing attacks at client side using auto-updated white-list,” EURASIP Journal on Information Security, vol. 2016, no. 1, pp. 1–11, 2016.
[6] F. Tajaddodianfar, J. W. Stokes, and A. Gururajan, “Exception: A character/word-level deep learning model for phishing url detection,” in ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2020, pp. 2857–2861.
[7] J. Seymour and P. Tully, “Generative models for spear phishing posts on social media,” arXiv preprint arXiv:1802.05196, 2018.
[8] Y. Huang, Q. Yang, J. Qin, and W. Wen, “Phishing url detection via cnn and attention-based hierarchical rnn,” in 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/18th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). IEEE, 2019, pp. 112–119.
[9] Y. Su, “Research on website phishing detection based on lstm rnn,” in 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), vol. 1. IEEE, 2020, pp. 284–288.
[10] D. Aksu, Z. Turgut, S. Üstebay, and M. A. Aydin, “Phishing analysis of websites using classification techniques,” in International Telecommunications Conference. Springer, 2019, pp. 251–258.
[11] A. O’Mara, I. Alsmadi, and A. AlEroud, “Generative adversarial analysis of phishing attacks on static and dynamic content of webpages,” in 2021 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking, 2021, pp. 1657–1662.
