Evasion Generative Adversarial Network for Low Data Regimes

Rizwan Hamid Randhawa, Nauman Aslam, *Member, IEEE*, Mohammad Alauthman, and Husnain Rafiq

***Abstract*—A myriad of recent literary works have leveraged generative adversarial networks (GANs) to generate unseen evasion samples. The purpose is to annex the generated data with the original train set for adversarial training to improve the detection performance of machine learning (ML) classifiers. The quality of generated adversarial samples relies on the adequacy of training data samples. However, in low data regimes like medical diagnostic imaging and cybersecurity, the anomaly samples are scarce in number. This paper proposes a novel GAN design called evasion generative adversarial network (EVAGAN) that is more suitable for low data regime problems that use oversampling for detection improvement of ML classifiers. EVAGAN not only can generate evasion samples but its discriminator can act as an evasion-aware classifier. We have considered auxiliary classifier GAN (ACGAN) as a benchmark to evaluate the performance of EVAGAN on cybersecurity (ISCX-2014, CIC-2017, and CIC2018) botnet and computer vision (MNIST) datasets. We demonstrate that EVAGAN outperforms ACGAN for unbalanced datasets with respect to detection performance, training stability, and time complexity. EVAGAN's generator quickly learns to generate the low sample class and hardens its discriminator simultaneously. In contrast to ML classifiers that require security hardening after being adversarially trained by GAN-generated data, EVAGAN renders it needless. The experimental analysis proves that EVAGAN is an efficient evasion hardened model for low data regimes for the selected cybersecurity and computer vision datasets.**

***Impact Statement*—Artificial Intelligence (AI) applications can help improve the quality of human life. The use of AI is not only limited to medical anomaly detection and drug discovery but can be leveraged in computer networks to keep people safe from malicious activities on the Internet. However, the AI-based models can be biased towards the majority class of data on which they are trained due to data imbalance. Anomaly data samples are always scarce as compared to the normal data samples. So this is an open research problem to solve. Our article is an effort to improve the AI-based methods in detection performance, stability and time complexity. Using the proposed technique, we can train our AI model using fewer anomaly samples, improving the cost-efficiency compared to the state-of-the-art in anomaly detection.**

***Index Terms*—Auxiliary classifier generative adversarial network (ACGAN), botnet, evasion generative adversarial network (EVAGAN), generative adversarial networks (GANs), low data regimes, MNIST.**

I. Introduction

Low data regimes are found in many real-life applications in which researchers face data scarcity problems [1]. The data scarcity pertains to the situation where one class is abundant in data samples (especially normal behavior) while the anomaly samples are rare and challenging to gather [2]. The data scarcity can also be described as a data imbalance problem potentially resulting in decision bias in machine learning (ML) classifiers. The network traffic datasets are one of the prime examples of data imbalance problems. Since the ML intrusion detection systems are data-hungry probabilistic models, having more data can improve their performance [3]. The real attacks can be emulated with dedicated machines in a lab environment using open-source operating systems like Kali Linux [4], [5]. However, there can be two main disadvantages of emulating real attacks: First, real data gathering can be expensive, involving multiple hardware resources like multiple computers and network switches [6]. Second, the emulated attacks may not accurately represent a real attack scenario. A cost-effective way of gathering the attacks' data is synthetic generation using AI generative models [7].

Synthetic data generation is also termed data oversampling. Using generative adversarial networks (GANs) as synthetic oversamplers has been a voguish research endeavor for low data regimes [3], [8]. Various researchers have demonstrated that GANs are more effective as compared to other synthetic oversamplers like SMOTE [2], [7], [9], [10]. It is found in numerous studies that due to the adversarial factor, GANs can better estimate the target probability distribution [2], [9], [11]. In a simple/vanilla GAN, two different neural networks generator ($\mathcal{G}$) and discriminator ($\mathcal{D}$) work antagonistically to learn from each other's experience to converge to Nash equilibrium [12]. As an oversampler, after being trained to a certain number of epochs, $\mathcal{G}$ is used to generate additional data. Depending on how well a GAN learned the input data probability distribution, the close resembling data is annexed to the original train set. This process is called data augmentation (DA), which many researchers have demonstrated to be effective in improving the detection performance of ML classifiers [13]–[17].

Manuscript received 19 January 2022; revised 21 March 2022 and 3 June 2022; accepted 9 July 2022. Date of publication 4 August 2022; date of current version 22 September 2023. This work was supported by the Research and Development Fund (RDF) of Northumbria University, Newcastle upon Tyne, U.K.). This paper was recommended for publication by Associate Editor Manuel Roveri upon evaluation of the reviewers' comments. *(Corresponding author: Rizwan Hamid Randhawa.)*

Rizwan Hamid Randhawa, Nauman Aslam, and Husnain Rafiq are with the Department of Computer and Information Sciences, Northumbria University, NE1 8ST Newcastle upon Tyne, U.K. (e-mail: rhr407@gmail.com; nauman.aslam@northumbria.ac.uk; husnain.rafiq@northumbria.ac.uk).

Mohammad Alauthman is with the Department of Information Security, University of Petra, Amman 11196, Jordan (e-mail: mohammad.alauthman@uop.edu.jo).

The code will be available at https://www.github.com/rhr407/EVAGAN., provided by the authors.

Digital Object Identifier 10.1109/TAI.2022.3196283

Since AI-based systems are prone to adversarial evasion attacks, it is imperative to harden the ML classifiers against adversarial evasions. Black box attackers can use GANs to generate evasion samples [15], [16], [18]. Therefore, employing GANs can be an effective technique to proactively design an adversarial aware classifier resulting from DA. Although DA is effective in helping the ML classifiers recognize the perturbed data samples, $\mathcal{D}$ of a GAN can be extended to act as a multiclass classifier so that it can be used as an anomaly detector [16], [19], [20]. In this way, we do not need to use DA as the $\mathcal{D}$ is trained simultaneously with $\mathcal{G}$. Auxiliary classifier GAN (ACGAN) is an example of such a GAN in which the $\mathcal{D}$ not only differentiates between fake and real samples but also can be used as a multiclass classifier [2], [21]. The advantage of extending the $\mathcal{D}$ in ACGAN is to improve training stability and quality of generated samples [21]. In this article, with the help of experimentation, we have demonstrated that ACGAN does not perform well in highly unbalanced datasets. So we propose a novel GAN based on ACGAN called evasion generative adversarial network (EVAGAN) that outperforms ACGAN in terms of detection performance, stability in training, and time complexity.

We summarize the main contributions of this article in the following aspects.

1) We propose a novel GAN model to design an evasion-aware discriminator as a sophisticated botnet detector.
2) We demonstrate by experiments that the existing use of ACGAN to design a sophisticated classifier can fail in highly unbalanced datasets.
3) We determine that EVAGAN outperforms ACGAN in terms of performance detection, stability, and time complexity for cybersecurity (CC) botnet and computer vision (CV) datasets.

Table I shows the main notations used in this article. The rest of this article has been organized as follows. Section II provides a comprehensive background of vanilla GANs, data oversampling, adversarial evasion, and ACGAN; Section III presents the details of the proposed model; Section IV gives a description of implementation details; Section V demonstrates the results; Section VI provides an analysis of the results; and Section VIII concludes this article.

II. Background

A. *Generative Adversarial Networks (GANs)*

A GAN combines two different neural networks, each having a unique structure. The one responsible for generating synthetic samples is called generator ($\mathcal{G}$), and the other that evaluates the generated samples is called discriminator ($\mathcal{D}$). Fig. 1 shows the block diagram of a classical/vanilla GAN. There are two consecutive steps in which a GAN is trained. In the first step, the $\mathcal{D}$ is trained on real data labeled as REAL, and the data generated by an untrained $\mathcal{G}$ is labeled as FAKE. In the next step, now that the $\mathcal{D}$ has trained already, it is tested on the fake data from $\mathcal{G}$, but this time intentionally labeled as REAL. The loss of the $\mathcal{D}$ on this falsely labeled data is fed back to the $\mathcal{G}$ which adjusts its weights in one complete batch training. There can be several batch iterations, after which one complete traversal of the dataset is complete, also known as an epoch. In the classical GAN, the generator model can be represented as $\mathcal{G}$: z$\rightarrow$ $\mathcal{X}$, where z is the normal distribution from noise space and $\mathcal{X}$ is the real data distribution.

TABLE I
Main Notations

Notation	Definition
$\mathcal{G}$	Generator
$\mathcal{D}$	Discriminator
z	Normal distribution from noise space
z	Noise samples
p_{data}	Probability distribution of real samples
p_z	Probability distribution of noise samples
$\mathcal{X}$	Real data distribution
$\mathbb{E}$	Expected value
c_m	Minority class labels
c_M	Majority class labels
y_{x_i}	Actual label of sample x_i in dataset $\mathcal{X}$

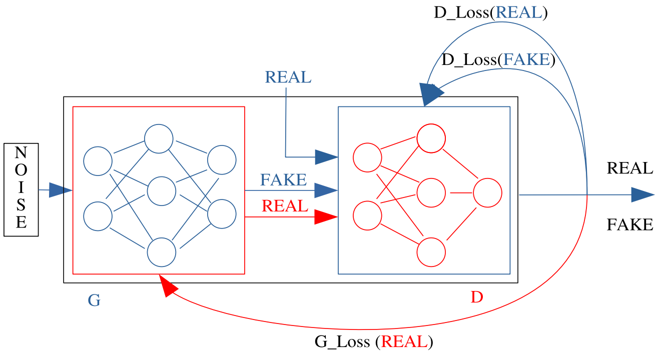

Fig. 1. Block diagram of a classical GAN.

The discriminator $\mathcal{D}$: $\mathcal{X}$ $\rightarrow$[0,1] model is a classifier that outputs an estimate of probability between 0 and 1 to mark whether the input data is real or fake. The objective function of the combined model can be represented by (1).

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{D},\mathcal{G}) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log \mathcal{D}(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-\mathcal{D}(\mathcal{G}(z)))]. \quad (1)$$

Here, $\mathbb{E}$ represents the expected value of the loss, and x and z denote the real and noise samples, respectively. At the same time, p_{data} and p_z are the probability distributions of real and noise data, respectively. The objective of a min–max game is to minimize the generator's loss in creating data resembling real data. Since the generator cannot control the loss of $\mathcal{D}$ on real data, still, it can maximize the loss of $\mathcal{D}$ on generated data $\mathcal{G}(z)$. The objective function of $\mathcal{G}$ is given by (2).

$$J^{\mathcal{G}}(\mathcal{G}) = \mathbb{E}_{z\sim p_z(z)}[\log(\mathcal{D}(\mathcal{G}(z)))] \quad (2)$$

As demonstrated in the Fig. 1, the losses of $\mathcal{D}$ on real $D\_Loss(REAL)$ and generated data $D\_Loss(FAKE)$, respectively, are fed to $\mathcal{D}$ using backpropagation. In the next step, in forward propagation, given label as REAL to the input generated samples (coming from $\mathcal{G}$), the evaluation is done by $\mathcal{D}$ and $G\_Loss(REAL)$ is fed back to $\mathcal{G}$ to update its weights. We call this step the combined model training. The combined model takes noise as input and the output of the $\mathcal{D}$ as the feedback to update the weights of the $\mathcal{G}$. This process keeps iterating till the number of epochs reaches a set value. The generator and discriminator do not learn further upon achieving the Nash equilibrium.

B. Data Oversampling and GANs

In low data regimes, oversampling or undersampling can help balance the datasets. However, undersampling might result in the loss of diversity. For oversampling, methods like SMOTE use the nearest neighbors and linear interpolation, which can be unsuitable for high-dimensional and complex probability distributions [9], [22]. Recent research works proposed algorithms for data oversampling. Kovács [23] compared 85 different oversampling techniques and suggested the three best-performing variants as SMOTE_IPF, ProWSyn, and polynom_fit_SMOTE. Randhawa et al. [7] have compared the performance of these three SMOTE variants with GANs. Through empirical results, they found that GANs outperform the three mentioned oversamplers in most of the adversarial training of ML classifiers.

C. ACGAN

ACGAN extends a classical GAN exploiting class labels in the training process [21]. Similar to a classical GAN, ACGAN includes two neural networks: a generator ($\mathcal{G}$) and a discriminator ($\mathcal{D}$). In addition to random noise samples z, the input of $\mathcal{G}$ includes class labels c. Therefore, the synthesized sample from $\mathcal{G}$ in ACGAN is $\mathcal{X}_{\text{fake}} = \mathcal{G}(c, z)$, instead of $\mathcal{X}_{\text{fake}} = \mathcal{G}(z)$. In other terms, ACGAN can generate the specified class data for which we feed labels to its $\mathcal{G}$. Simultaneously, the $\mathcal{D}$ of ACGAN works as a dual classifier for differentiating between the real/fake data and different classes of the input samples, whether coming from the real source or the $\mathcal{G}$.

The objective function of ACGAN consists of two parts: The first is the log-likelihood L_S of the correct source data and the second is the log-likelihood L_C of the real class labels. $\mathcal{D}$ is trained to maximize $L_C + L_S$ and $\mathcal{G}$ learns to maximize $L_C - L_S$. In other words, the objective of $\mathcal{D}$ is to improve the two likelihoods, while the goal of $\mathcal{G}$ is to assist $\mathcal{D}$ in improving the performance on class label discrimination. $\mathcal{G}$ will also try to suppress the log-likelihood of $\mathcal{D}$ on fake samples. The $\mathcal{D}$ outputs both a probability distribution over sources and the class labels, respectively, $[P(S|\mathcal{X}), P(C|\mathcal{X})] = \mathcal{D}(\mathcal{X})$, where S are the sources (real/fake) and C are the class labels. Equations (3) and (4) denote the L_s and L_c, respectively.

$$\begin{aligned} L_S &= \mathbb{E}[\log P(S = \text{real}|\mathcal{X}_{\text{real}})] \\ &\quad + \mathbb{E}[\log P(S = \text{fake}|\mathcal{X}_{\text{fake}})] \qquad (3) \\ L_C &= \mathbb{E}[\log P(C = c|\mathcal{X}_{\text{real}})] \\ &\quad + \mathbb{E}[\log P(C = c|\mathcal{X}_{\text{fake}})]. \qquad (4) \end{aligned}$$

A careful observation of Fig. 2 suggests that there seems to be no tremendous difference between ACGAN and EVAGAN; however, the significance of simple modifications in the generator input, discriminator output, and loss functions is discussed in more detail in Section III.

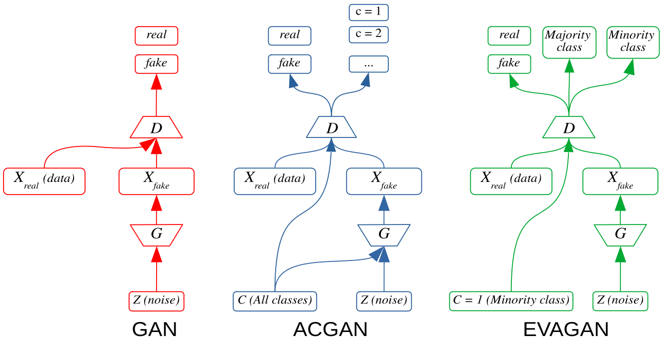

Fig. 2. Comparison of EVAGAN model with vanilla GAN and ACGAN.

D. Adversarial Evasion and GANs

The decision bias in ML classifiers can lead to the misclassification of malicious samples as normal. The attackers can exploit this intrinsic nature of ML classifiers to incarnate evasion samples, particularly in low data regimes. The adversarial evasion j* is a perturbed version of an input sample j such that the j* = j + η, where η is a carefully crafted perturbation. When making an adversarial attack, η could be sought and selected so that the classifier can not discriminate the j* from j [24], [25]. The researchers usually employ adversarial training to make the classifiers proactively aware of the evasion samples. However, this is not needed if we use the $\mathcal{D}$ of a GAN as a classifier to differentiate not only between the fake and real samples but also between normal and anomaly samples. The fake samples generated by the $\mathcal{G}$ are also learnt at the same time, so it is better to consider the power of $\mathcal{D}$ as an evasion-aware classifier. We do not need to use extra ML classifiers, which is a common practice in various literary works, to design such a classifier [19], [20].

To this end, we propose EVAGAN that provides such type of $\mathcal{D}$ and compare its performance with the $\mathcal{D}$ of ACGAN and other ML classifiers, xgboost (XGB), decision tree (DT), Naive Bayes (NB), random forests (RF), logistic regression (LR), and k-nearest neighbors (KNN). Following rigorous experimentation, we explore that EVAGAN's $\mathcal{D}$ not only outperforms the ML classifiers in black box testing but also gives 100% accuracy in normal and evasion samples estimation. The details of the experimental results are discussed in Section VI.

III. EVAGAN

In this section, we discuss the motivation behind the design of EVAGAN, the structural explanation of its generator and discriminator, along with the objective and loss functions.

A. Motivation

Considering the generator ($\mathcal{G}$) of ACGAN, $\mathcal{X}_{\text{fake}} = \mathcal{G}(c, z)$, where c is the class label and $\mathcal{G}$ has to generate the samples of all classes. Hence, the number of the samples generated by $\mathcal{G}$ may include $C = \{c_1, c_2, c_3, \ldots, c_n\}$, which may not be a

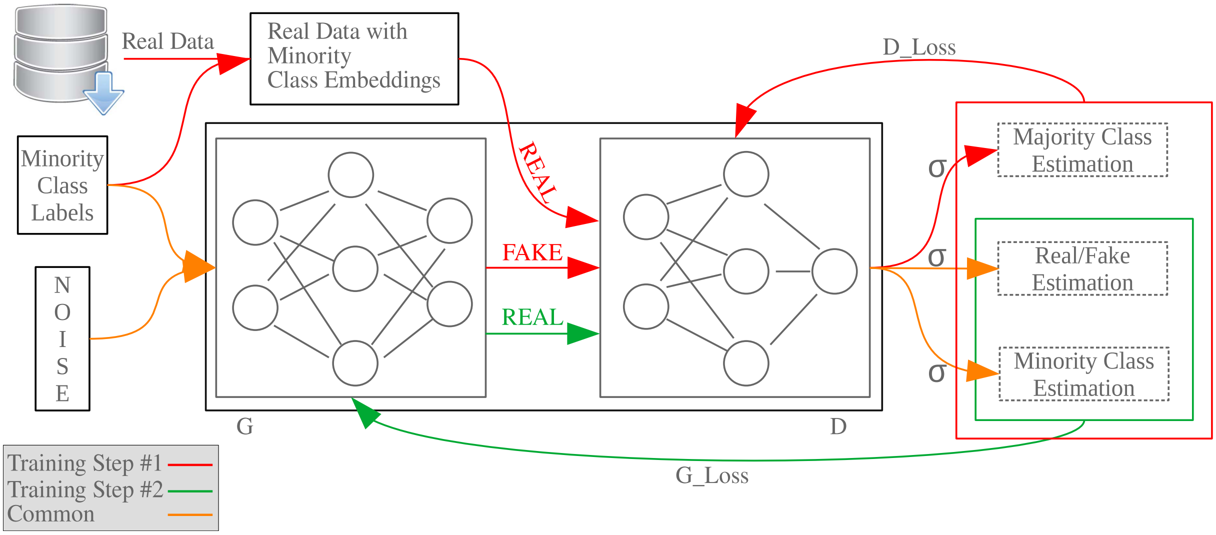

Fig. 3. EVAGAN architecture.

requirement in low data regimes. Since we only need to generate a low sample class with labels c_m instead of all the classes, so the generator does not need to be aware of the classification performance of $\mathcal{D}$ on majority class samples. In this way, the training time of $\mathcal{G}$ is reduced as the diversity seen by the $\mathcal{G}$ is less complex to generate a single class sample. Due to this reason, we can not only improve the performance of the $\mathcal{G}$ but also can harden the $\mathcal{D}$ simultaneously with fewer c_m samples. The ratio of the different class labels can vary the performance of $\mathcal{G}$ as this is a stochastic process. However, in most cases, the normal class samples will be more than the anomaly samples. Note that EVAGAN design is dedicated to binary class problems where the samples of a minority class are scanty. For using EVAGAN for multiclass cases, each anomaly class should be considered separately from the normal class to make it a binary classification problem. However, the concept can be extended to multiclass, which we leave to future work.

B. Architecture

The design of EVAGAN is inspired by ACGAN as we want to develop a classifier model that hardens itself on the GAN-generated evasion samples. The main structure of EVAGAN consists of two neural networks: the generator $\mathcal{G}$ and the discriminator $\mathcal{D}$. In contrast to ACGAN, EVAGAN's model is limited to labels from a single class embedded with noise as the input to the generator ($\mathcal{G}$). The details of the $\mathcal{G}$ are explained in Section III-C. Fig. 3 shows the detailed architecture of EVAGAN. There are three types of color-graded arrows shown in the figure. The red-colored arrows demonstrate the first training step in which only $\mathcal{D}$ is trained. The green arrows depict the second training step for the $\mathcal{G}$. The orange arrows show the involvement of common inputs (minority class labels and noise) and outputs (real/fake and minority class estimations) for both training steps mentioned previously. These two steps of a typical GAN training were expressed in Section II-A. The discriminator $\mathcal{D}$ of EVAGAN has three different outputs for the estimation of majority, minority, and fake/real classes. Sigmoid functions have been used for the three outputs, each with binary cross-entropy (BCE) loss. The details of $\mathcal{D}$ are further expressed in Section III-D. The loss functions are also mentioned in respective subsections of the $\mathcal{G}$ and $\mathcal{D}$.

Fig. 3 shows a red outlined box on the right side to illustrate the three different probability estimations as outputs from $\mathcal{D}$. These three estimations are used to compute the loss of $\mathcal{D}$ in the first step of EVAGAN training. A green outlined box, including the real/fake estimation and minority class estimation, computes the G_Loss to be fed back to the $\mathcal{G}$ in the backpropagation of the combined model training (second step of EVAGAN training). Note that the output of the $\mathcal{D}$ is distributed using three different sigmoid units to separate the probabilities of each class, i.e., the majority, sources (real/fake), and minority. The majority and minority class estimations could be combined using a single sigmoid function. However, keeping them separate has three advantages. The first is to avoid the loss of the majority class being fed back to the $\mathcal{G}$. Second, it simplifies the model with no extra training cost. Third, we can conveniently separate the predictions for the test set samples, which is discussed in Section V.

C. Generator

The generator ($\mathcal{G}$) of EVAGAN only takes noise n and the single class labels $c = 1$. The labels are embedded in the input layer of the $\mathcal{G}$. The objective function of the $\mathcal{G}$ has two parts, as shown in (5) and (6).

$$I^{\mathcal{G}}(\mathcal{G}) = \mathbb{E}_{z\sim p_z(z)}[\log(\mathcal{D}(\mathcal{G}(z)))] \tag{5}$$

$$J^{\mathcal{G}}(\mathcal{G}) = \mathbb{E}_{c_m\sim y_m}[\log P(C = c_m|\mathcal{X}_{m_{\text{fake}}})] \tag{6}$$

Equation (5) is the objective function of $\mathcal{G}$ similar to (2). The goal is to minimize the log-likelihood of the fake samples being classified as fake by $\mathcal{D}$. In (6), $J^{\mathcal{G}}(\mathcal{G})$ is the objective function of $\mathcal{G}$ for improving the log-likelihood of minority class samples coming from the $\mathcal{G}$ into the $\mathcal{D}$. Here, y_m denotes the minority class label in the real dataset, and P is the output probability from $\mathcal{D}$. Since the $\mathcal{G}$ only needs to generate c_m samples, so it should only receive the loss of $\mathcal{D}$ on the estimation of minority class and the sources, i.e., the samples being real or fake. The objective function of $\mathcal{G}$ is to maximize the $\mathcal{D}$ loss on the fake source. At the same time, it will assist in minimizing the $\mathcal{D}$ loss on c_m samples. Equation (7) shows the objective function of $\mathcal{G}$.

$$L^{\mathcal{G}}(\mathcal{G}) = J^{\mathcal{G}}(\mathcal{G}) - I^{\mathcal{G}}(\mathcal{G}). \tag{7}$$

The cross-entropy (CE) loss of two different probability distributions $p(x)$ and $q(x)$ can be denoted using (8), where x denotes the samples belonging to the $\mathcal{X}$ dataset.

$$CE(p,q) = -\sum_{x \epsilon X} p(x) \log q(x). \tag{8}$$

Let y_{x_i} be the actual label of sample x_i in dataset $\mathcal{X}$, $P(S = \text{fake}|\mathcal{X}_{m_{\text{fake}}})$ be the predicted probability distribution of generated samples being fake and $P(C = c_m|\mathcal{X}_{m_{\text{fake}}})$ be the predicted probability distribution from $\mathcal{D}$ for minority class labels c_m, then the loss function of $\mathcal{G}$ for N samples will be given by (9).

$$\begin{aligned} G\_Loss = & -\frac{1}{N}\sum_{i=1}^{N}[y_{x_i}^{\text{fake}}(\log P(S = \text{fake}|\mathcal{X}_{m_{\text{fake}}})) \\ & + y_{x_i}^{c_m}(1 - \log P(C = c_m|\mathcal{X}_{m_{\text{fake}}}))]. \end{aligned} \tag{9}$$

In (9), $y_{x_i}^{\text{fake}}$ and $y_{x_i}^{c_m}$ are the actual labels for fake and minority classes, respectively. According to (9), the goal of $\mathcal{G}$ is to minimize the G_Loss, so it tends to reduce the correct estimation of $\mathcal{D}$ on fake samples by suppressing the term $\log P(S = \text{fake}|\mathcal{X}_{m_{\text{fake}}})$. For the second objective, it will try to increase the value of $\log P(C = c_m|\mathcal{X}_{m_{\text{fake}}})$ so that the second term in the equation can also be suppressed in value.

D. Discriminator

For the $\mathcal{D}$ model of EVAGAN, we have separated the majority and minority class estimations using two different sigmoid (σ) functions as demonstrated in Fig. 3. The benefit of separating the majority and minority class estimations is that we can feedback only minority class estimation to the $\mathcal{G}$. The other advantage of this structure is that we can separately calculate the estimation of both classes on test datasets to compare it with the ACGAN model later done in Section V. The objective function of $\mathcal{D}$ has three parts as given by (10), (11), and (12). For the minority class terminologies, we use "m," and for the majority class, we use "M" in the following equations:

$$L_M = \mathbb{E}_{c_M \sim y_{M_{\text{real}}}}[\log P(C = c_M|\mathcal{X}_{M_{\text{real}}})] \tag{10}$$

$$\begin{aligned} L_{S_m} = & \mathbb{E}_{y_{m_{\text{real}}}}[logP(S = real|\mathcal{X}_{m_{\text{real}}})] \\ & + \mathbb{E}_{y_{m_{\text{fake}}}}[logP(S = fake|\mathcal{X}_{m_{\text{fake}}})] \end{aligned} \tag{11}$$

$$\begin{aligned} L_m = & \mathbb{E}_{c_m \sim y_{m_{\text{real}}}}[\log P(C = c_m|\mathcal{X}_{m_{\text{real}}})] \\ & + \mathbb{E}_{c_m \sim y_{m_{\text{fake}}}}[\log P(C = c_m|\mathcal{X}_{m_{\text{fake}}})]. \end{aligned} \tag{12}$$

The first goal of the $\mathcal{D}$ is to correctly estimate the majority class distribution from the real samples only as $\mathcal{G}$ does not generate the majority class samples. Equation (10) denotes the log-likelihood for the real majority class samples. Equation (11) represents the source log-likelihood for the real and fake minority class samples. Equation (12) summarizes the real and fake log-likelihoods from the $\mathcal{D}$ for minority class samples. Hence, the objective function of the $\mathcal{D}$ can be represented as the sum of the three log-likelihoods to be maximized by the $\mathcal{D}$ as given by (13)

$$L^{\mathcal{D}}(\mathcal{D}) = L_M + L_{S_m} + L_m. \tag{13}$$

The loss function of $\mathcal{D}$ can be derived using (5) and (6), given by (14).

$$\begin{aligned} D\_Loss = & -\frac{1}{N}\sum_{i=1}^{N}[y_{x_i}^{c_M}(\log P(S = c_M|\mathcal{X}_{M_{\text{real}}})) \\ & + y_{x_i}^{\text{real}}(\log P(S = \text{real}|\mathcal{X}_{m_{\text{real}}})) \\ & + (1 - y_{x_i}^{\text{real}})(1 - \log P(S = \text{real}|\mathcal{X}_{m_{\text{real}}})) \\ & + y_{x_i}^{c_{m_{\text{real}}}}(\log P(C = c_m|\mathcal{X}_{m_{\text{real}}})) \\ & + (1 - y_{x_i}^{c_{m_{\text{real}}}})(1 - \log P(C = c_m|\mathcal{X}_{m_{\text{real}}}))]. \end{aligned} \tag{14}$$

In (14), the loss of $\mathcal{D}$ has been derived from three different BCE losses for majority class, sources, and minority class estimations. Note that we have ignored the loss on c_M for being fake because no majority class samples are being generated by the $\mathcal{G}$.

IV. Implementation Details

A. Experimental Setup

The experiments were performed on a GPU workstation, AMD Ryzen threadripper 1950x with a 16-core processor and GeForce GTC 1070 Ti (8 GB) graphics card, running ubuntu 20.04. Keras, TensorFlow, Sklearn, and Numpy libraries were used in the Jupyter notebook and visual studio code (VSCode). The source code of EVAGAN has been provided on GitHub under MIT license.[1]

B. Data Preparation

For experimentation, we have used CC botnet and CV MNIST datasets. The quantitative analysis of EVAGAN was performed on CC datasets. We have followed the work done by Randhawa et al. [7] for dataset selection of botnet. We have used three datasets, ISCX-2014, CIC-2017, and CIC-2018, from the Canadian Institute of Cybersecurity (CIC). The features were extracted using a utility called CICFlowMeter-v4 provided by the CIC. We have inherited the same feature set as mentioned in [7]. The reader may refer to this article for more details on the feature set used for the three datasets. The number of samples of benign vs. botnet is mentioned in Table II. The qualitative analysis was performed using visual inspection. For this purpose, we used the MNIST handwriting digits dataset.

[1] [Online]. Available: https://github.com/rhr407/EVAGAN.

TABLE II
DISTRIBUTION OF NORMAL AND BOTNET SAMPLES IN CYBERSECURITY BOTNET DATASETS

Dataset	Normal	Real_bots	Total samples
ISCX-2014	246929	Virut: 1748	248677
CIC-IDS2017	70374	Ares: 1956	72330
CIC-IDS2018	390961	Ares/Zeus: 2560	393521

C. CC Datasets

Following is the detail of CC datasets and the botnet samples used in this article. This subsection also includes the preprocessing methodology for the selected datasets.

1) ISCX-2014 Dataset: The ISCX-2014 dataset [26] is a combination of three publicly available datasets ISOT [27], ISCX 2012 IDS [28], and CTU-13 [29]. As per the ISCX website's details, it complies with generality, realism, and representativeness. The generality represents the richness of diversity of botnet behavior. Realism can be defined as the closeness with the actual traffic captured, and representativeness is the ability to reflect the real environment, which a botnet detector would need in deployment. Only the Virut botnet was selected for this article because it had fewer samples than other botnets except for Zeus, which had insufficiently low samples. The labels of SMTP or NSIS were not available on the website.[2] Hence, we used a subset with all the normal traffic flows and Virut samples. In this way, we could use this dataset as a good example of an unbalanced set. The distribution of the normal and Virut samples is shown in Table II.

2) CIC-IDS2017 Dataset: The botnet chosen for the CIC-IDS2017 was Ares. For this bot, the traffic was collected on Friday, July 7, 2017, from 10:02 A.M. to 11:02 A.M. in the CIC facility. The dataset is available on the CIC website.[3] Similar to ISCX-2014, a subset of this dataset using all the normal flows with the selected botnets was created. The ratio of the number of samples is mentioned in Table II.

3) CIC-IDS2018 Dataset: To create another subset of an unbalanced dataset for analysis, we used CIC-IDS2018. This dataset included samples for Ares and Zeus botnets. We created a subset of all the normal and 2560 botnet traffic flows to generate another unbalanced dataset.

4) Feature Selection: The quality of a botnet dataset determines the performance of the botnet detectors in general and the number of distinct features in particular. A reduced feature set may not perform a stronger classification as compared to an enhanced set of nonredundant features [7]. Beigi et al. [26] summarized the most important network flow features that could be helpful in botnet detection. We have used almost all of these features, which were mentioned in [30] as well. The CICFlowMeter-v4 utility was used to extract 80 flow and time-based features[4] from their *.pcap* files. This utility can be advantageous for extracting the mentioned features for any input *.pcap* file.

5) Preprocessing: The ISCX-2014 dataset has not been labeled to be used in ML-based experiments. We used the information provided on the CIC website for IPs associated with the particular botnets to label the dataset. After labeling, we performed preprocessing; all the high and low skewed values were removed to suppress outliers. The columns with NaN, Inf, and zero standard deviation were removed. Finally, the dataset was scaled to the [0,1] range to use rectified linear unit (ReLU) activation function in the GAN model for data generation. The CIC-IDS2017 and CIC-IDS2018 were already labeled, so we only did preprocess for these two datasets after extracting the unbalanced subsets. Our experiments used 70% of the cybersecurity subsets as training sets, and the rest of the 30% was used for testing the models and ML classifiers.

D. CV Dataset

1) MNIST Dataset: The MNIST dataset is a simplified collection of handwritten digits ranging from 0 to 9 for training and testing various ML algorithms [31]. The purpose of using this dataset was to evaluate the performance of EVAGAN against ACGAN in terms of the visual quality of the images generated in balanced and unbalanced scenarios.

E. Model Comparison of EVAGAN With ACGAN

For comparison, we constructed four different variants of GANs, respectively, ACGAN_CC, EVAGAN_CC, ACGAN_CV, and EVAGAN_CV. ACGAN_CC and EVAGAN_CC were trained and tested on CC datasets, and ACGAN_CV and EVAGAN_CV used CV datasets. The implementation details of each version in terms of hyperparameters can be found in Table III.

1) ACGAN_CC and EVAGAN_CC: The structure of ACGAN_CC and EVAGAN_CC was made up of densely connected feed-forward neural network (FFNN) for both $\mathcal{G}$ and $\mathcal{D}$. The activation functions in hidden layers for both GANs were ReLUs. The hidden layers were regularized using batch normalization, and the optimizer type was Adam with BCE. The difference between ACGAN_CC and EVAGAN_CC is in the output layers of $\mathcal{D}$. The $\mathcal{D}$ of ACGAN_CC outputs two neurons, one for the source probability and the other for the class probability for two classes (normal and botnet). The activation function is sigmoid for both outputs. The output layer structure of EVAGAN_CC has three neurons, one for the normal class, the second for the source probability, and the third for the botnet class (minority class). Each of the three outputs leverages the sigmoid as the activation function.

2) ACGAN_CV and EVAGAN_CV: The CV-based GAN architecture is different as it deals with image data compared to tabular data in CC. We need to use the convolutional neural network (CNN) instead of FFNN with other layers specific for image generation or detection. The output layer of $\mathcal{D}$ is similar to CC-based GAN implementations, except the Adam optimizer's loss function has BCE for source estimations and sparse categorical cross-entropy (SCCE) for class labels. Here, BCE could have been used; however, minimal changes to the code were made to maintain the integrity of the original ACGAN.

[2] [Online] Available: https://www.unb.ca/cic/datasets/botnet.html.
[3] [Online] Available: https://www.unb.ca/cic/datasets/ids-2017.html.
[4] [Online] Available: https://www.unb.ca/cic/datasets/ids-2018.html.

TABLE III
CC AND CV GAN MODELS

Parameter	ACGAN_CC	EVAGAN_CC	ACGAN_CV	EVAGAN_CV
Network type	FFNN		CNN	
Number of layers	$\mathcal{G}$: 5, $\mathcal{D}$: 5		$\mathcal{G}$: 2, $\mathcal{D}$: 3	
Activations	$\mathcal{G}$: ReLU, $\mathcal{D}$: LeakyReLU (output: sigmoid)		$\mathcal{G}$: ReLU (output: tanh), $\mathcal{D}$: ReLU (output: sigmoid, softmax)	
Batch size (b)	256			
Neurons in input layer	$\mathcal{G}$: latent dimension, class label vector size, $\mathcal{D}$: feature size			
Neurons in layer 1	$\mathcal{G}$: 32, $\mathcal{D}$: 128		$\mathcal{G}$: 128, $\mathcal{D}$: 32	$\mathcal{G}$: 128, $\mathcal{D}$: 32
Neurons in layer 2	$\mathcal{G}$: 64, $\mathcal{D}$: 64		$\mathcal{G}$: 64, $\mathcal{D}$: 64	
Neurons in layer 3	$\mathcal{G}$: 128, $\mathcal{D}$: 32		$\mathcal{D}$: 128	
Neurons in output layer	$\mathcal{G}$: feature size, $\mathcal{D}$: 2	$\mathcal{G}$: feature size, $\mathcal{D}$: 3	$\mathcal{G}$: feature size, $\mathcal{D}$: 2	$\mathcal{G}$: feature size, $\mathcal{D}$: 3
Layer regularization	$\mathcal{G}, \mathcal{D}$: $BatchNorm$			
Optimizer	Adam (beta_1=0.0002, beta_2=0.5)			
Loss function	BCE		BCE, SCCE	
Learning rate	5e-4			
Epochs	150			

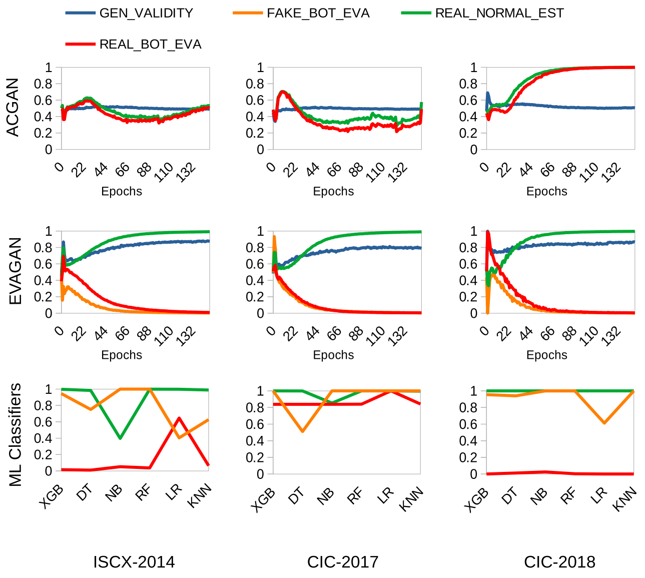

Fig. 4. CC estimations: The estimations on test data and data generated by the relative GANs along with the results of six different ML-classifiers.

However, in ACGAN_CC, we have used BCE as we converted the CNN-based code to FFNN ourselves. In this way, we could keep ACGAN_CC and EVAGAN_CC as similar as possible for a fair comparison.

V. Results

This section shows the results of the GAN implementations around two types of datasets: CC and CV GANs.

A. CC GANs

The results for quantitative analysis of the $\mathcal{D}$'s performance on generated samples validity (GEN_VALIDITY), fake/generated botnet samples evasion (FAKE_BOT_EVA), real normal/majority class estimation (REAL_NORMAL_EST), and real botnet/minority class evasion (REAL_BOT_EVA) are demonstrated in Fig. 4. The ML classifier results are been shown in this figure for the three CC datasets for comparison. Equations (15)–(18) represent the mathematical expressions for these performance indicators. We have used Keras $model.predict$ function to compute the values where the $model$ is $\mathcal{D}$ as our prime objective is to devise an intelligent evasion aware classifier. Following is a brief detail of each evaluation parameter.

1) GEN_VALIDITY: In (15), $\hat{\mathcal{G}}(z, c_m)[0]$ denotes the predicted value for the source being fake or real. The Keras $model.predict$ function outputs an array, so the average of the first elements in the array will be the source validity of the generated samples after every epoch. The more this value is close to "1," the more it will be regarded as real.

$$GEN\_VALIDITY = \frac{\sum[\hat{\mathcal{G}}(z, c_m)[0]]}{N}. \tag{15}$$

2) FAKE_BOT_EVA: In (16), $\hat{\mathcal{G}}(z, c\_m)[1]$ represents the probability estimation of generated minority/botnet class samples. Since the label for minority/botnet class is "0," so ideally, we expect the model to output a value close to "0." We represent this estimation as the evasion of the generated samples. So the more this value is close to "0," the less evasion will be. Note that this is the second value in the sum of the $model.predict$ function output.

$$FAKE\_BOT\_EVA = \frac{\sum[\hat{\mathcal{G}}(z, c_m)[1]]}{N}. \tag{16}$$

In (17), $\hat{\mathcal{X}}_{\text{normal}_{\text{test}}}[2]$ represents the probability estimation of majority/normal class samples. Since the majority/normal class label is "1," ideally, we expect the model to output a value close to "1." Note that this is the third value in the sum of the $model.predict$ function output for the normal samples from the test set.

$$REAL\_NORMAL\_EST = \frac{\sum[\hat{\mathcal{X}}_{\text{normal}_{\text{test}}}[2]]}{N}. \tag{17}$$

3) REAL_BOT_EVA: In (18), $\hat{\mathcal{X}}_{\text{botnet}_{\text{test}}}[1]$ represents the probability estimation of the real minority/botnet class samples. Our expectation from the model is to output the value close to "0," similar to FAKE_BOT_EVA. This is the second value in the sum of the $model.predict$ function output for the botnet samples from the test set.

$$REAL\_BOT\_EVA = \frac{\sum[\hat{\mathcal{X}}_{\text{botnet}_{\text{test}}}[1]]}{N}. \tag{18}$$

4) Losses: The losses of $\mathcal{D}$ for real and fake minority classes and majority/normal class and the loss of $\mathcal{G}$ are demonstrated in Fig. 5 for both ACGAN and EVAGAN.

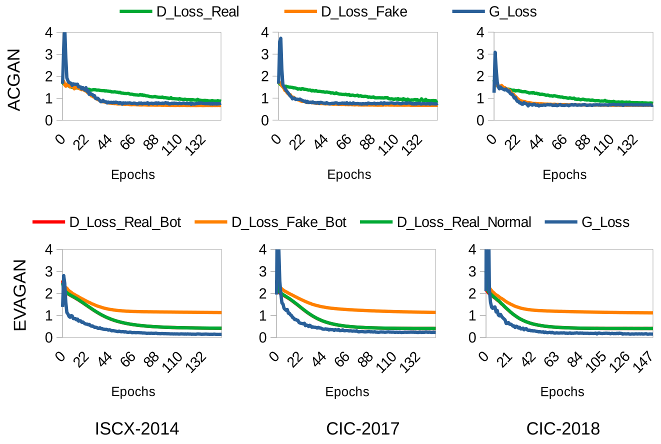

Fig. 5. CC GANs losses: The training losses for ACGAN and EVAGAN on three different CC datasets.

B. CV GANs

For ACGAN_CV and EVAGAN_CV, we use MNIST handwritten digits dataset. Only two classes of digits, “0” and “1,” were used in ACGAN_CV, as due to SCCE, its model does not accept fewer than two classes. For ECAGAN_CV, we use only the “0” digit as the minority class. Since the MNIST data is already balanced, we need to undersample the values of the “0” digit class to demonstrate the difference in performance. Four different undersampling levels are devised in Section VI. The evaluation parameters were equivalent to those used in CC GANs. For instance, GEN_Validity is the same as GEN_VALIDITY; GEN_Eva is similar to FAKE_BOT_EVA with the minority class from MNIST, i.e., “0” in our case. Similarly, ONE_Est is equivalent to REAL_NORMAL_EST, and ZERO_Eva is comparable to REAL_BOT_EVA in CC GANs. Fig. 6 demonstrates the quantitative results for the four undersampling scenarios. Note that out of four, the first scenario exhibits 0% undersampling. There are three scenarios with undersampling, 50%, 90%, and 99%. For qualitative analysis, the output from $\mathcal{G}$s of both ACGAN_CV and EVAGAN_CV is demonstrated in Figs. 8 and 9. These results are also based on the undersampling cases.

VI. Performance Comparison of EVAGAN With ACGAN

A. Detection Performance

In Fig. 4, for the ACGAN_CC, the values for the REAL_NORMAL_EST and REAL_BOT_EVA remain close to each other. This implies that the $\mathcal{D}$ of ACGAN_CC is not able to discriminate between the majority and minority classes well due to the imbalance problem in all the three CC datasets. The $\mathcal{D}$ of ACGAN_CC remains confused for the two classes in ISCX-2014 and CIC-2017 datasets. For the majority class, ACGAN_CC performs equally well as EVAGAN_CC for CIC-2018 (shown in the second row of Fig. 4). However, due to the small number, it regards the minority class samples as the majority class instances. The second row in Fig. 4 shows the results of EVAGAN_CC for the estimations on the test set. It can be observed that as compared to ACGAN_CC, the $\mathcal{D}$ of EVAGAN_CC perfectly differentiates between the majority and minority classes and, after each epoch, tends to improve its detection performance for all the three CC datasets.

We have used FAKE_BOT_EVA as an indicator of evasion awareness of the $\mathcal{D}$ in the case of EVAGAN_CC only because ACGAN_CC generates two classes of data, so the $\mathcal{G}$ of ACGAN_CC would generate a random number of samples from both classes leading to nondeterministic values of FAKE_BOT_EVA. However, we compare the performance of this metric with ML classifiers. The last row of Fig. 4 shows the results of the six different ML classifiers for the values of the majority, minority, and generated class samples. It can be inferred that EVAGAN_CC tends to outperform the ML classifiers for all three values after a certain number of epochs. The ML classifiers for black-box testing perform worst in the case of FAKE_BOT_EVA as compared to EVAGAN_CC for all the three CC datasets. This implies that the $\mathcal{D}$ of EVAGAN_CC is not only adept at discriminating between real minority samples but can also easily detect the fake minority samples that ML classifiers are not good at discerning. Another significant advantage of this $\mathcal{D}$ is that we do not need to employ ML classifiers in CC for learning adversarial evasion. Researchers use GANs to generate adversarial samples to be augmented with the training set for retraining ML classifiers to make them adversarially aware. In the case of EVAGAN_CC, we save that time as the $\mathcal{D}$ classifier/detector model is trained alongside the GAN training.

It can be further illustrated from Fig. 4 that the value of GEN_VALIDITY in the case of ACGAN_CC seems to remain close to 0.5 for all the three CC datasets. It means that the $\mathcal{D}$ is confused in deciding whether the generated samples from $\mathcal{G}$ are real or fake. However, in the case of EVAGAN_CC, for all the three datasets, $\mathcal{G}$’s performance is improving with each epoch. This implies that $\mathcal{D}$ is being fooled and still learning, while in the case of ACGAN_CC, the $\mathcal{D}$ has already been saturated because $\mathcal{G}$ is not generating new samples that can fool $\mathcal{D}$.

1) CV GANs: Fig. 6 demonstrated the results of different undersampling scenarios to mimic the low data regimes for the MNIST dataset. Note that the detection performance of the $\mathcal{D}$ for both ACGAN_CV and EVAGAN_CV for the majority and minority classes remains ideal from the very start. The reason is that, unlike CC datasets, the CV dataset has many strong features due to which $\mathcal{D}$ is easily able to differentiate between the “0” digit and “1” digit samples. However, the effect of undersampling can be seen for the minority class or digit “0” data. In contrast, the $\mathcal{D}$ of EVAGAN_CV seems to be smart enough to give steady values for all the undersampling cases, especially for minority class evasion (as depicted in red color lines). Due to the sufficient number of samples, the majority class should be detected easily by both GANs. However, in the case of 99% undersampling, ACGAN_CV exhibits a poor performance even detecting this class. For GEN_Validity (represented by the blue lines), Fig. 6 shows that in undersampling cases, the performance of $\mathcal{G}$ of ACGAN_CV deteriorates in the worst manner and does not show any useful pattern of learning. This implies that in low data regimes, $\mathcal{G}$ is not performing any better as compared to EVAGAN_CV. However, EVAGAN_CV also shows the deterioration in $\mathcal{G}$’s performance, but that is not as phenomenal as that of ACGAN_CV.

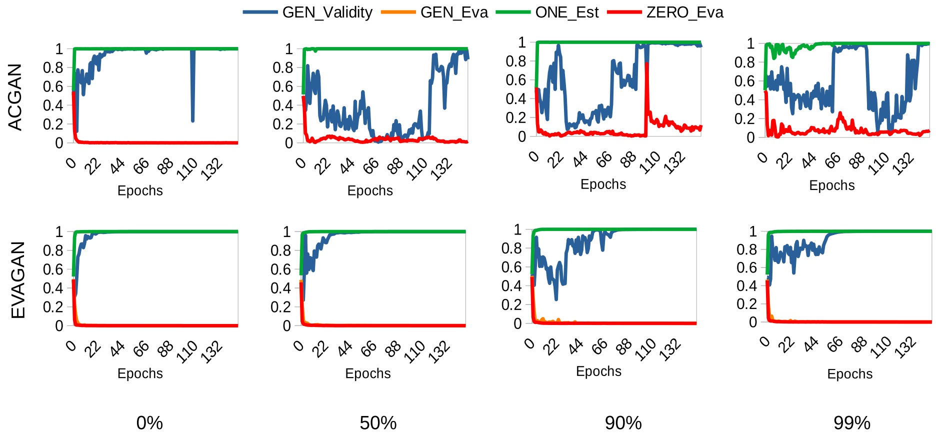

Fig. 6. CV GANs estimations: The estimations on the test set for ACGAN and EVAGN for MNIST dataset in different undersampling scenarios. The range of the estimation value on the *y*-axis is from 0 to 1.

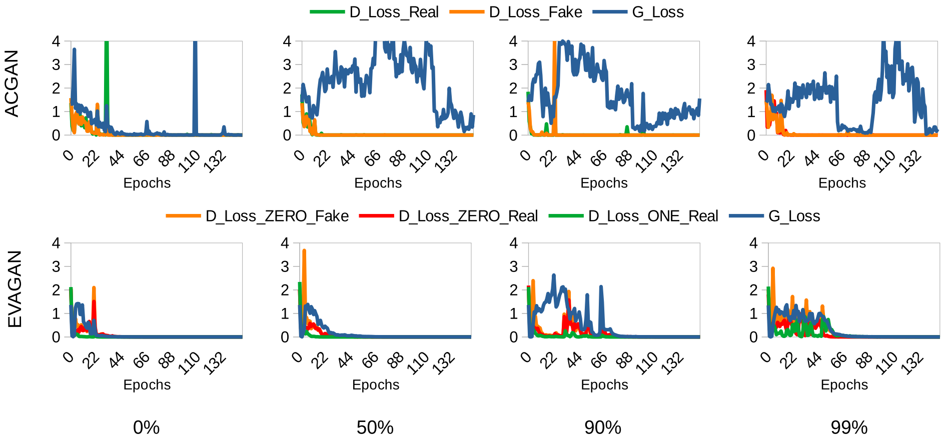

Fig. 7. CV GANs losses: The training losses on train set for ACGAN and EVAGN for MNIST dataset in different undersampling scenarios. The upper limit to the loss has been fixed to 4 for the sake of consistency to highlight the difference.

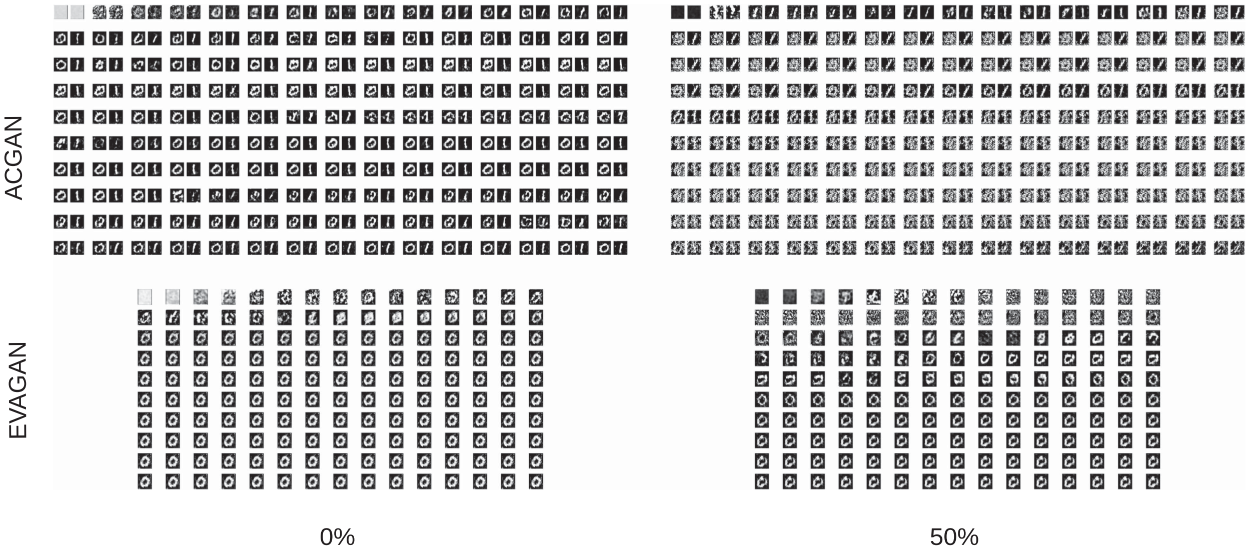

Fig. 8. Qualitative analysis of $\mathcal{G}$ for CV GANs with 0% and 50% undersampling of “0” digit class.

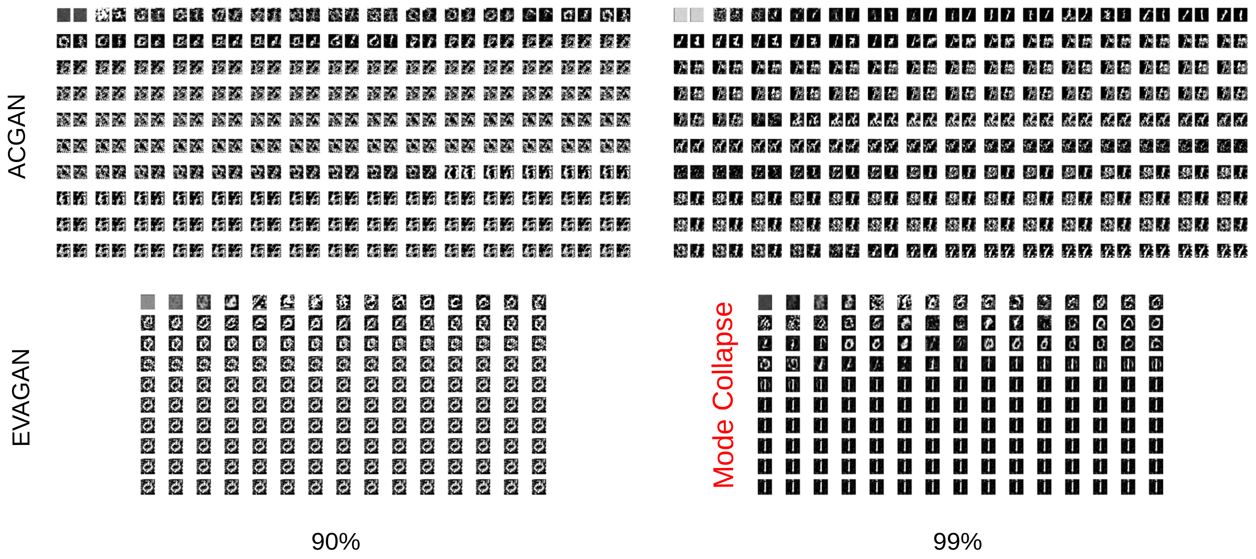

Fig. 9. Qualitative analysis of $\mathcal{G}$ for CV GANs with 90% and 99% undersampling of "0" digit class.

B. Stability

Fig. 5 shows the $\mathcal{D}$ and $\mathcal{G}$ losses for CC GANs. It can be inferred from this diagram that the values for all the losses seem to be converging. This shows that the GANs are saturating toward Nash equilibrium. However, in the case of EVAGAN_CC, the losses tend to be more steady with each epoch and achieve the lowest point sooner than ACGAN_CC. Similarly, for CV GANs, the EVAGAN_CV losses in all the undersampling cases tend to be more stable as compared to ACGAN_CV, as demonstrated in Fig. 7.

C. Qualitative Performance

It is nontrivial to demonstrate the performance of a GAN in the case of CC datasets [32]. Since we can not visualize the generated network traffic, we need to validate the EVAGAN with the help of CV datasets. The rationale for using CV datasets is that if EVAGAN outperforms ACGAN in unbalanced scenarios, it would be equally acceptable for CC datasets. Since our purpose is not to generate quality traffic for CC, we need to design an evasion-aware anomaly detector. So, evaluating EVAGAN_CC for quality traffic generation is not within the scope of this article.

The previously mentioned undersampling scenarios for CV GANs are demonstrated in Figs. 8 and 9. There are two 15×10 matrices of pictures in each figure. The number of images in each matrix equals the total number of epochs, i.e., 150. In each figure, the upper row belongs to the ACGAN_CV output of the $\mathcal{G}$ and the lower row corresponds to the output from $\mathcal{G}$ of EVAGAN_CV. Note that for ACGAN_CV, there are two classes being output from $\mathcal{G}$, and for EVAGAN_CV, only one "0" digit class is generated. For the undersampling scenario, which contains 50% fewer "0" class samples, the deterioration for ACGAN_CV starts getting evident, but EVAGAN_CV can generate "0" digits. For the case of 90% undersampling, the ACGAN_CV quality further deteriorates; however, EVAGAN_CV is still generating the "0" class samples, although slightly faded. As expected, in the 99% undersampling case, the ACGAN_CV is still struggling to generate the minority class digit "0," but an interesting case has happened for EVAGAN_CV. Since the number of samples is minuscule so the feedback taken from the $\mathcal{D}$ by $\mathcal{G}$, on some accidentally generated "1" digit, gave a small value of G_Loss. Due to this reason, the $\mathcal{G}$ started generating the majority class "1" digit after epoch 47. This situation is called a mode collapse, an inherent problem in GANs. However, we can infer that EVAGAN_CV may not perform well in a highly unbalanced scenario. This is an interesting research direction to investigate further using other CV datasets. On the other hand, ACGAN_CV is also stuck in mode collapse after epoch 140, where in place of class "0," the "1" class samples start appearing. However, in the case of EVAGAN_CV, despite mode collapse, the generated samples from class "1" are of higher quality, which means that its $\mathcal{G}$ is more powerful as compared to that of ACGAN in highly unbalanced scenarios.

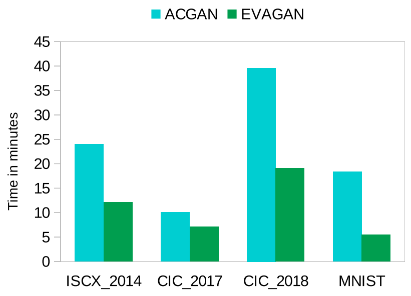

Fig. 10. Time complexity.

D. Time Complexity

The time complexity bar chart is demonstrated in Fig. 10, where the y-axis represents the values of the training time in minutes. The MNIST dataset case with no undersampling was used to compare the results. The time complexity may vary on

TABLE IV
COMPARISON WITH PEER WORKS

Paper	**Addressing evasion problem**	**Adversarial training/ augmentation**	**Low data regime**	**Architecture**	**Datasets used**	**Maximum accuracy**
ID-GAN [19]	✗	✗	✓	Multiclass ACGAN	NSL-KDD	83.10%
G-IDS [3]	✗	✓	✓	Vanilla GAN	NSL-KDD	–
AE-CGAN [33]	✗	✓	✓	Auto encoder with conditional GAN	CIC-2017	100%
[34]	✓	✓	✗	ANN, CNN, RNN	UNSW-NB15, NSL-KDD	97%
Attack-GAN [35]	✓	✓	✗	Sequence GAN	CTU-13	–
GADoT [36]	✓	✓	✗	WGAN-GP	Custom-SYN, Scapy-SYN, CICIDS2017, UNB201X	–
DIGFuPAS [37]	✓	✓	✓	WGAN	CICIDS2017	–
min-max Training [46]	✓	✓	✓	DNN	NSL-KDD	93.4%
attackGAN [47]	✓	✗	✓	WGAN	NSL-KDD	–
CVAE-AN [38]	✓	✓	✓	Conditional VAE and GAN	CICIDS2017	98%
CEGAN [48]	✗	✓	✓	CNN	MNIST, EMNIST, F-MNIST CIFAR-10, CINIC-10	96.48%
EVAGAN	✓	✗	✓	Binary class ACGAN	ISCX-2014, CIC-2017, CIC-2018, MNIST	100%

different platforms (for instance, Google Colab); however, the plot in Fig. 10 shows the results on the workstation that we have used (as mentioned in Section IV). It can be observed that EVAGAN always takes less time than its counterpart for all four datasets. The reason lies in the notion that the $\mathcal{G}$ of EVAGAN in the cases of all the datasets needs to follow lesser diversity as compared to ACGAN. Although the batch size of 256 (given in Table III) is the same for both GANs, the amount of time taken by EVAGAN is always less. Due to the stochastic nature of the input noise z for the $\mathcal{G}$, we cannot estimate the exact time in minutes for every training cycle; however, the average time of EVAGAN always remains less as compared to ACGAN.

A question might arise why we did not make ACGAN generate only the minority class samples. The answer to this question is that we would have to make changes in the structure of both $\mathcal{G}$ and $\mathcal{D}$ along with the loss functions. The SCCE loss does not allow us to use less than two classes, so we need to use BCE loss with other structural modifications. EVAGAN is the name of this transformation.

VII. Comparison of EVAGAN With Peer Techniques

The EVAGAN model is an enhanced version of ACGAN, dedicated to low data regimes for learning adversarial evasion examples generated during GAN training. So the most suitable existing model for the comparison can be ACGAN, the details of which have been explained in Section VI. However, this section mentions peer techniques similar to EVAGAN that indirectly address the adversarial evasion problem. We have summarized the comparison in the following subsections and in Table IV.

A. Data Augmentation

There are several techniques both in CV and CS that propose the DA for enhancing the ML classifiers' performance [3], [15], [33]–[38]. However, EVAGAN itself acts as a powerful adversarial evasion-aware model in which the discriminator ($\mathcal{D}$) acts as a classifier. So there is no need to generate evasion samples from a GAN model, augment with the training set, and then train a separate ML classifier. This property of EVAGAN makes it superior to all the techniques based on DA in terms of time complexity.

B. Computer Vision vs. Cybersecurity Low Data Regimes

There are plenty of works that address the problem of low data regimes [39]–[45]. However, their datasets and model architectures differ from those used in this article. We have mentioned a few in Table IV. Since the ML studies are biased towards data, experimenting with other datasets can be a potential future work.

C. Architecture Comparison

Yin et al. [19] proposed a model in which the discriminator is acting as a multiclass classifier; however, their work is not destined toward adversarial evasion generation in low data regimes as they have considered the normal class samples to train the generator of their GAN. Our article differs in a way that we do not feed our generator ($\mathcal{G}$) with normal class samples, which makes the $\mathcal{G}$'s job easier. This saves the training time and improves the estimation accuracy of malicious samples, even being scanty.

D. Accuracy and Time Complexity

EVAGAN produces ideal results of estimation for both majority and minority classes, as high as 100% for all the datasets used, as mentioned in Section V. The comparison has been provided with ACGAN; however, the accuracy in comparison with other similar models is at par as well. The accuracy values determined from the literature for some other models addressing similar problems are given in Table IV. The time complexity as compared to the ACGAN model is discussed in Section V; however, it would be nontrivial to compare with other peer models in respect of training time as the model architecture and hyperparameters vary enormously. The experiments for EVAGAN and ACGAN were performed on the same machine as mentioned in Section IV, so we claim the time complexity comparison with ACGAN only.

VIII. Conclusion

Adversarial evasion attacks on AI-based systems are a portending threat that needs to be dealt with using intuitive methods. Adversarial learning is one of the modern techniques to make ML classifiers proactively adept at detecting adversarial evasion samples. This article proposes a novel GAN model called EVAGAN that generates adversarial evasions in low data regimes. EVAGAN is an enhancement of a well-known model called ACGAN. EVAGAN aims to design an adversarial-aware classifier for anomaly detection. We have used two datasets: one from the cybersecurity domain for botnets and the other from the computer vision called MNIST. EVAGAN's discriminator is superior to ACGAN in terms of detection performance, stability, and time complexity. At the same time, the qualitative analysis shows that EVAGAN outperforms ACGAN in unbalanced scenarios. EVAGAN model has been designed for binary classification problems.

Further investigation for multiclass design is a potential research direction. Experiments with other datasets would be highly desirable to further evaluate EVAGAN for the said parameters. For the qualitative analysis, handwritten digits other than "0" and "1" could be used to validate EVAGAN's superiority over ACGAN. A comparison with few-shot learning could be an interesting research direction.

References

[1] M. Moret, L. Friedrich, F. Grisoni, D. Merk, and G. Schneider, "Generative molecular design in low data regimes," *Nature Mach. Intell.*, vol. 2, no. 3, pp. 171–180, 2020.

[2] L. Vu and Q. U. Nguyen, "Handling imbalanced data in intrusion detection systems using generative adversarial networks," *J. Res. Develop. Inf. Commun. Technol.*, vol. 2020, no. 1, pp. 1–13, 2020.

[3] M. H. Shahriar, N. I. Haque, M. A. Rahman, and M. Alonso, "G-ids: Generative adversarial networks assisted intrusion detection system," in *Proc. IEEE 44th Annu. Comput., Softw., Appl. Conf.*, 2020, pp. 376–385.

[4] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. Int. Conf. Inf. Syst. Secur. Privacy*, 2018, pp. 108–116.

[5] A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features," in *Proc. Int. Conf. Inf. Syst. Secur. Privacy*, 2017, pp. 253–262.

[6] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-iot dataset," *Future Gener. Comput. Syst.*, vol. 100, pp. 779–796, 2019.

[7] R. H. Randhawa, N. Aslam, M. Alauthman, H. Rafiq, and F. Comeau, "Security hardening of botnet detectors using generative adversarial networks," *IEEE Access*, vol. 9, pp. 78276–78292, 2021.

[8] V. Sushko, J. Gall, and A. Khoreva, "One-shot GAN: Learning to generate samples from single images and videos," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 2596–2600.

[9] J. Engelmann and S. Lessmann, "Conditional Wasserstein GAN-based oversampling of tabular data for imbalanced learning," 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417421000233

[10] G. Kovács, "Smote-variants: A python implementation of 85 minority oversampling techniques," *Neurocomputing*, vol. 366, pp. 352–354, 2019.

[11] H. Ba, "Improving detection of credit card fraudulent transactions using generative adversarial networks," 2019, *arXiv:1907.03355v1*.

[12] I. Goodfellow et al., "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[13] A. Antoniou, A. Storkey, and H. Edwards, "Data augmentation generative adversarial networks," 2017, *arXiv:1711.04340*.

[14] T. Merino et al., "Expansion of cyber attack data from unbalanced datasets using generative adversarial networks," in *Proc. Int. Conf. Softw. Eng. Res., Manage. Appl.*, 2019, pp. 131–145.

[15] M. Usama et al., "Generative adversarial networks for launching and thwarting adversarial attacks on network intrusion detection systems," in *Proc. 15th Int. Wireless Commun. Mobile Comput. Conf.*, 2019, pp. 78–83.

[16] Z. Lin, Y. Shi, and Z. Xue, "IDSGAN: Generative adversarial networks for attack generation against intrusion detection," *Adv. Knowl. Discovery Data Mining*, J. Gama, T. Li, Y. Yu, E. Chen, Y. Zheng, and F. Teng, Eds. Cham: Springer International Publishing, pp. 79–91, 2022.

[17] H. Zhang, X. Yu, P. Ren, C. Luo, and G. Min, "Deep adversarial learning in intrusion detection: A data augmentation enhanced framework," 2019, *arXiv:1901.07949*.

[18] Q. Yan, M. Wang, W. Huang, X. Luo, and F. R. Yu, "Automatically synthesizing dos attack traces using generative adversarial networks," *Int. J. Mach. Learn. Cybern.*, vol. 10, no. 12, pp. 3387–3396, 2019.

[19] C. Yin, Y. Zhu, S. Liu, J. Fei, and H. Zhang, "Enhancing network intrusion detection classifiers using supervised adversarial training," *J. Supercomputing*, vol. 76, no. 9, pp. 6690–6719, 2019.

[20] Y. Yin and C. Zhu, "An enhancing framework for botnet detection using generative adversarial networks," in *Proc. Int. Conf. Artif. Intell. Big Data*, 2018, pp. 228–234.

[21] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier GANs," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2642–2651.

[22] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002.

[23] G. Kovács, "An empirical comparison and evaluation of minority oversampling techniques on a large number of imbalanced datasets," *Appl. Soft Comput.*, vol. 83, 2019, Art. no. 105662.

[24] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2017, pp. 506–519.

[25] G. Apruzzese, M. Andreolini, M. Marchetti, A. Venturi, and M. Colajanni, "Deep reinforcement adversarial learning against botnet evasion attacks," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 4, pp. 1975–1987, Dec. 2020.

[26] E. B. Beigi, H. H. Jazi, N. Stakhanova, and A. A. Ghorbani, "Towards effective feature selection in machine learning-based botnet detection approaches," in *Proc. IEEE Conf. Commun. Netw. Secur.*, 2014, pp. 247–255.

[27] D. Zhao et al., "Botnet detection based on traffic behavior analysis and flow intervals," *Comput. Secur.*, vol. 39, pp. 2–16, 2013.

[28] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Comput. Secur.*, vol. 31, no. 3, pp. 357–374, 2012.

[29] S. Garcia, "Malware capture facility project," *cvut*, 2013.

[30] R. K. Sharma, H. K. Kalita, and B. Issac, "Are machine learning based intrusion detection system always secure? An insight into tampered learning," *J. Intell. Fuzzy Syst.*, vol. 35, no. 3, pp. 3635–3651, 2018.

[31] Y. LeCun, "The MNIST database of handwritten digits," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[32] Y. Guo, G. Xiong, Z. Li, J. Shi, M. Cui, and G. Gou, "TA-GAN: GAN based traffic augmentation for imbalanced network traffic classification," in *Proc. Int. Joint Conf. Neural Netw.*, 2021, pp. 1–8.

[33] J. Lee and K. Park, "AE-CGAN model based high performance network intrusion detection system," *Appl. Sci.*, vol. 9, no. 20, 2019, Art. no. 4221.

[34] R. Abou Khamis and A. Matrawy, "Evaluation of adversarial training on different types of neural networks in deep learning-based IDSS," in *Proc. Int. Symp. Netw., Comput. Commun.*, 2020, pp. 1–6.

[35] Q. Cheng, S. Zhou, Y. Shen, D. Kong, and C. Wu, "Packet-level adversarial network traffic crafting using sequence generative adversarial networks," 2021, *arXiv:2103.04794*.

[36] M. Abdelaty, S. Scott-Hayward, R. Doriguzzi-Corin, and D. Siracusa, "GADoT: GAN-based adversarial training for robust DDoS attack detection," in *Proc. 9th IEEE Conf. Commun. Netw. Secur.*, 2021, pp. 119–127.

[37] C. P. X. Qui et al., "Strengthening IDS against evasion attacks with GAN-based adversarial samples in SDN-enabled network," in *Proc. RIVF Int. Conf. Comput. Commun. Technol.*, 2021, pp. 1–6.

[38] U. Sabeel, S. S. Heydari, K. Elgazzar, and K. El-Khatib, "CVAE-AN: Atypical attack flow detection using incremental adversarial learning," in *Proc. IEEE Glob. Commun. Conf.*, 2021, pp. 1–6.

[39] W. Li, X. Zhong, H. Shao, B. Cai, and X. Yang, "Multi-mode data augmentation and fault diagnosis of rotating machinery using modified ACGAN designed with new framework," *Adv. Eng. Inform.*, vol. 52, 2022, Art. no. 101552.

[40] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan, "Gan-based synthetic medical image augmentation for increased CNN performance in liver lesion classification," *Neurocomputing*, vol. 321, pp. 321–331, 2018.

[41] X. Yi, E. Walia, and P. Babyn, "Generative adversarial network in medical imaging: A review," *Med. Image Anal.*, vol. 58, 2019, Art. no. 101552.

[42] A. Waheed, M. Goyal, D. Gupta, A. Khanna, F. Al-Turjman, and P. R. Pinheiro, "COVIDGAN: Data augmentation using auxiliary classifier GAN for improved COVID-19 detection," *IEEE Access*, vol. 8, pp. 91916–91923, 2020.

[43] V. Chinbat and S.-H. Bae, "Ga3n: Generative adversarial autoaugment network," *Pattern Recognit.*, vol. 127, 2022, Art. no. 108637.

[44] W. Jo and D. Kim, "OBGAN: Minority oversampling near borderline with generative adversarial networks," *Expert Syst. with Appl.*, vol. 197, 2022, Art. no. 116694.

[45] A. Madhu and S. K, "EnvGAN: A GAN-based augmentation to improve environmental sound classification," *Artif. Intell. Rev.*, pp. 1–20, Feb. 2022. [Online]. Available: https://doi.org/10.1007/s10462-022-10153-0

[46] S. Grierson, C. Thomson, P. Papadopoulos, and B. Buchanan, "Min–max training: Adversarially robust learning models for network intrusion detection systems," in *Proc. 14th Int. Conf. Secur. Inf. Netw.*, 2021, pp. 1–8.

[47] S. Zhao, J. Li, J. Wang, Z. Zhang, L. Zhu, and Y. Zhang, "attackGAN: Adversarial attack against black-box IDS using generative adversarial networks," *Procedia Comput. Sci.*, vol. 187, pp. 128–133, 2021.

[48] S. Suh, H. Lee, P. Lukowicz, and Y. O. Lee, "CeGAN: Classification enhancement generative adversarial networks for unraveling data imbalance problems," *Neural Netw.*, vol. 133, pp. 69–86, 2021.

Rizwan Hamid Randhawa received the B.S. degree in electronic engineering from International Islamic University Islamabad, Islamabad, Pakistan, in 2008 and the Master's degree in computer science from Information Technology University, Lahore, Pakistan, 2017. He is currently working toward the Ph.D. degree in computer science with Northumbria University, Newcastle upon Tyne, U.K.

He has vast experience with embedded systems in Pakistan's private and public sector organisations. His research interests include AI-based botnet detection, IoT security, and embedded systems design and development for IoT platforms.

Nauman Aslam (Member, IEEE) received the Ph.D. degree in engineering mathematics from Dalhousie University, Halifax, NS, Canada, in 2008.

He is currently a Professor with the Department of Computer and Information Science, Northumbria University, Newcastle upon Tyne, U.K., where he leads the Network Systems and Security research group. He was an Assistant Professor with Dalhousie University. He has authored or coauthored more than 100 papers in peer-reviewed journals and conferences. His current research focuses on addressing wireless body area networks and IoT, network security, QoS-aware communication in industrial wireless sensor networks, and artificial intelligence application in communication networks. His research interests include diverse but interconnected areas related to communication networks.

Mohammad Alauthman received the B.Sc. degree in computer science from Hashemite University, Zarqa, Jordan, in 2002, the M.Sc. degree in computer science from Amman Arab University, Amman, Jordan, in 2004, and the Ph.D. degree in network security and botnet detection from Northumbria University, Newcastle upon Tyne, U.K., in 2016.

He is currently an Assistant Professor with the Information Security Department, Petra University, Amman, Jordan. His research interests include cybersecurity, cyber forensics, advanced machine learning, and data science applications.

Husnain Rafiq received the B.S. and M.S. degrees in computer science from the Capital University of Science and Technology, Islamabad, Pakistan, in 2015 and 2017, respectively. He is currently working toward the Ph.D. degree in android malware detection with Northumbria University, Newcastle upon Tyne, U.K.

From 2015 to 2018, he was a Junior Lecturer with the Capital University of Science and Technology. His research interests include information security and forensics, machine learning, and malware analysis.