Model-Agnostic Generation-Enhanced Technology for Few-Shot Intrusion Detection

Junpeng He[1], Lingfeng Yao[1], Xiong Li[1*], Muhammad Khurram Khan[2], Weina Niu[1], Xiaosong Zhang[1], Fagen Li[1]

[1*]University of Electronic Science and Technology of China, Chengdu, 611731, China.
[2]Center of Excellence in Information Assurance (CoEIA), King Saud University, Street, Riyadh, 11653, State, Kingdom of Saudi Arabia.

*Corresponding author(s). E-mail(s): lixiong84@gmail.com;
Contributing authors: junpenghe@std.uestc.edu.cn; 202122080537@std.uestc.edu.cn; mkhurram@KSU.EDU.SA; niuweina1@126.com; johnsonzxs@uestc.edu.cn; fagenli@uestc.edu.cn;

Abstract

Malicious traffic on the Internet has become an increasingly serious problem, and several artificial intelligence (AI)-based malicious traffic detection methods have been proposed. Generally, AI-based methods need numerous benign and specific types of malicious traffic training instances to achieve better detection results. However, for attacks with only a few instances, known as the few-shot attacks, these methods often perform poorly, and how to train a model for detecting few-shot attacks is a huge challenge. For this problem, we propose a novel intrusion detection system based on generative adversarial networks and model-agnostic meta-learning. The system adopts a hybrid detection mechanism where an anomaly-based classifier determines whether incoming traffic is malicious and a signature-based classifier identifies the class of malicious traffic. In the system, the samples of few-shot attacks are augmented by maximizing the use of meta-knowledge and then applied to assist the detection of few-shot attacks to obtain better detection results. The experiments show that for CSE-CIC-IDS2018 and Bot-IoT datasets, this system can detect malicious traffic with 94.3%/1.8% TPR/FPR and 99.8%/0.1% TPR/FPR, respectively, and also can identify the class of the few-shot attacks with 95.2% and 91.9% accuracy, respectively. Compared with other related methods, the system improves the accuracy of identifying few-shot attacks on these two datasets by at least 2.2% and 1.5%, respectively. Additionally, a parameter visualization process is designed, which shows the fast-adaptive property and better generalization capability of the system.

Keywords: Intrusion detection, model-agnostic meta-learning, generative adversarial network, few-shot attacks, imbalanced dataset

1 Introduction

Nowadays, the improving network communication frequency contributes to the increasing amount of intrusions on the Internet, especially in sixth-generation (6G) networks and the public Internet of Things (IoT) [1]. As a consequence, devices, servers and all kinds of infrastructure are more vulnerable. By using maliciously crafted code, an adversary can execute these intrusions to eavesdrop or interrupt the normal working status of a target machine [2]. In order to detect intrusions, a rule-based network intrusion detection system (NIDS) was first proposed, which can automatically monitor network traffic packets and traces based on a series of pre-defined rules [3]. However, in rule-based NIDS, the design of rules is usually difficult. In addition, the rule database needs to be continuously updated to deal with new intrusions, which brings non-negligible maintenance costs.

In order to overcome the shortcomings of the rule-based mechanism, several artificial intelligence (AI)-based NIDSes have been proposed. These AI-based NIDSes learn detection knowledge based on ready-made traffic features, which are then applied to infer new incoming network traffic. Although the traffic features need to be labeled manually or automatically, AI-based NIDSes do have lower maintenance costs than rule-based ones, which require continuous manual iteration. These NIDSes have applied various AI methods. As for the categories of these NIDSes, some adopt traditional machine learning (ML) methods such as support vector machine (SVM) [4] and random forest (RF) [5], others utilize deep learning (DL) methods like deep neural network (DNN) [6]. They perform well on some intrusion classification tasks where enough training samples are available, such as the identification Denial-of-Service (DoS)/Distributed Denial-of-Service (DDoS) and botnet attacks. Nevertheless, it is difficult to obtain sample instances of some intrusions (e.g., SQL Injection [7] and Zero-day [8]) in real environments. These intrusions are known as "few-shot attacks", while other intrusions are named "non-few-shot attacks". If a dataset contains these few-shot attacks, it may possess an imbalanced construction where the proportion of some attack labels is significantly small. An AI-based NIDS trained on this dataset largely fails to classify these attacks due to the lack of samples. This problem is known as the imbalanced dataset.

A feasible way to solve this problem is to augment instances for the few-shot attacks. The generative adversarial network (GAN) is a commonly used tool to do so. Generally, a GAN consists of two models, and they play a game against each other to generate instances that fit a target data distribution [9]. Our previous work [10] has shown that GAN can be used to generate adversarial attacks to bypass and enhance NIDSes. However, the training process of GAN still needs sufficient samples to form a reasonable target data distribution, which conflicts with the reality of few-shot attacks.

Meta-learning is another available technique for few-shot attack detection, since it can rapidly transfer the meta-knowledge learned from the non-few-shot traffic into the tasks for detecting the few-shot attacks [11]. As a typical form of meta-learning, model-agnostic meta-learning (MAML) learns the optimized biased initial model parameters as meta-knowledge [12]. Due to its knowledge transfer characteristics [13, 14], MAML is applied to NIDS and shows great performance on the few-shot attacks. Unfortunately, MAML-based NIDSes may not be able to infer new few-shot attacks with good generalization ability because the limited training samples are usually inadequate to reflect the real data distribution. No matter how excellent the meta-learning is, the detection of few-shot attacks still lacks better generalization in the imbalanced datasets.

To solve the aforementioned challenges, we designed a Model-Agnostic Generation-Enhanced Technology-based NIDS, named MAGET. It generates instances for few-shot attacks and utilizes the generated instances to assist few-shot intrusion detection. It includes a GAN-based generation process and a hybrid detection mechanism. The contributions of this paper are as follows:

- A novel knowledge transfer approach called MAGET is proposed. This approach is intended to realize few-shot intrusion generation to assist with few-shot attack classification. The generation part draws on the idea of GAN, while both the generation and classification parts are MAML-based.
- A visual variation of model parameters is carefully designed and discussed. It explains the differences and benefits of MAGET on fast-adaptive property and better generalization capability compared with the original MAML and GAN theoretically.
- The MAGET outperforms baseline and advanced methods on few-shot attacks in signature-based classification, as demonstrated by extensive experiments.

The remainder of this paper is structured as follows. Section 2 introduces the background knowledge and the related works in this domain. Section 3 explains the threat model and the system design of MAGET. Section 4 describes the workflow of MAGET. The experimental details are given in Section 5. Section 6 provides the parameter visualization. Conclusion and future works are finally provided in Section 7.

2 Background and Related Works

This section introduces some background on intrusion detection and the used MAML and GAN. Besides, the related works on few-shot attack classification are reviewed in this section.

2.1 Background of intrusion detection

Intrusion detection technology was first proposed by Jim Anderson [15], and now it has become an important cybersecurity guardian juxtaposed with the firewall. Generally, intrusion detection technology can be divided into several types according to different criteria. Among them, network intrusion detection is a popular one, which passively deploys nodes at gateways, routers or relays. These nodes, known as NIDSes, can inspect incoming traffic and detect potential behaviors that sniff or destroy host systems [16].

Basically, NIDSes are categorized into anomaly-based and signature-based [17]. The former defines a set of normal behaviors based on historical records, and the system will treat the input as "anomaly" if it is not in the normal range. The latter defines a set of rules that associate the suspicious events with the concrete attack types, and the system classifies current input based on these rules. Generally, anomaly-based classification is better at sniffing unknown intrusions, but may cause high false positive rates, while signature-based classification is effective at identifying rule-related attacks, but unable to detect unknown malicious behaviors. Additionally, there is a hybrid method that combines these two types of classification [18], which usually starts with an anomaly classification and then associates the anomaly behaviors with the corresponding signatures.

2.2 Background of MAML and GAN

MAML is one of the meta-learning strategies with quick concept adaptability. A model is first trained in the meta-training step based on prior knowledge of a batch of training tasks $T_{k=1}^{K}$, where the prior knowledge is the initialized model parameters Φ' in MAML, and then the model is used for solving the problem T_{test} in a meta-testing step [12]. More formally, as illustrated in Equation (1), the goal of MAML is to find an optimal Φ' such that the base learner can finish a new task as quickly as possible.

$$\Phi' * = arg \min_{\Phi'} E_{T_k \sim P(T_k)}[\zeta_{T_k}(O(D_{T_k}, \zeta_{T_k}|\Phi'))] \tag{1}$$

where T_k is a task selected from the probability distribution of tasks $p(T)$. $O(D_{T_k}, \zeta_{T_k}|\Phi')$ is an optimization procedure that utilizes predefined Φ', dataset and loss function from T_k. This equation outputs updated weights that perform well on T_i.

GAN is another famous technique that introduces game theory in neural network training for feature generation [19]. Its architecture includes two DL-based learners called generator (G) and discriminator (Dis). The objective of G is to generate instances to bypass Dis whereas Dis is to judge whether the input instances are from G or a real dataset. The overall loss function of GAN is shown in Equation (2), which consists of two objective functions. The first one encourages Dis to maximize $log(Dis(x))$ whereas the second one motivates G to minimize $log(1 - Dis(G(z)))$. In this equation, x is the instance sampled from the distribution of the real dataset P_{data} and z is the noise vector sampled from a defined distribution $P(\cdot)$ (e.g., the Gaussian distribution). G can generate different instances of the same label with different z. The training process of these two models is adversarial due to their objectives. Ideally, G can generate authentic instances that fit P_{data} and confuse Dis after training.

$$\begin{aligned}\zeta_{\text{GAN}} &= \zeta_{Dis} + \zeta_G \\ &= E_{x \sim P_{data}(x)}[log(Dis(x))] + \\ &\quad E_{z \sim P(z)}[log(1 - Dis(G(z)))]\end{aligned} \tag{2}$$

2.3 Related works of few-shot attack classification

The typical studies on few-shot attack classification are summarized in Table 1. Basically, these studies are classified into meta-learning-based methods that maximize the meta-knowledge utilization, data augmentation methods that enrich the imbalanced datasets, and hybrid methods. I_1 means that the target label is the real few-shot attack type, whose samples are hard to obtain in reality. I_2 denotes the usage of training optimization for models with few-shot samples. I_3 symbolizes showing the classification results of every single few-shot attack. I_4 denotes the usage of data augmentation. I_5 refers to the process explanation of the proposed methods.

For meta-learning-based methods, Xu et al.[20] proposed an FC-Net based on meta-learning to detect the visualized network traffic. They claim that their method performs well on the sub-datasets that they constructed from the public datasets, but the attack labels utilized in their experiment are not real few-shot labels, which possess enough samples in the public datasets. The real few-shot labels that have only a few samples in the public datasets were not considered (e.g., SQL Injection). Liang et al.[21] realized an optimized intra/inter-class-structure-based variational few-shot learning model to overcome the specific out-of-distribution problem, so as to detect the attacks in two imbalanced datasets more effectively. However, the performances on single few-shot subcategories were not measured. Concretely, their experiment combined four subcategories of DoS in CICIDS2017 [22] into one label and detected NSL-KDD [23] at the level of four main categories. Other works [24][25][26] classified few-shot attacks through a few training sets. However, their detection results only show the overall performance instead of the classification details of specific few-shot attacks.

For data augmentation methods, SMOTE [27] and GAN [9][28] are utilized to append the few-shot attack samples. Nevertheless, the generation process of these methods still relies on the limited training samples, which may not reflect the real data distribution of few-shot attacks. Consequently, these methods may be unstable. Furthermore, for the GAN-based data augmentation method, its parameters may converge at the local optimum due to the lack of generalization ability.

Table 1 Summary approaches for few-shot detection

Approaches	Models	I_1	I_2	I_3	I_4	I_5
Meta learning	FC-Net [20]	○	●	◐	○	○
	OICS-VFSL [21]	◐	●	○	○	○
	FSL IDS [24]	◐	●	◐	○	○
	FS-IDS [25]	●	●	◐	○	○
	FSL-Capsule [26]	◐	●	○	○	○
Data augmentation	SMOTE [27]	◐	○	◐	●	○
	GAN [9]	●	○	●	●	○
	IGAN-IDS [28]	◐	●	○	●	○
Hybrid	FAML [29]	○	●	○	●	○
	MAGET	●	●	●	●	●

"○" shows that the method does not pay attention to the issue,
"●" shows that the method fully pays attention to the issue,
"◐" shows that the method partly pays attention to the issue.

Additionally, FAML [29] uses MAML to achieve GAN-based few-shot image generation based on the meta-knowledge of other abundant images. Its generation process is more fluent and gets excellent results than directly training GAN with few-shot samples. However, it is for image generation rather than intrusion detection.

Overall, no literature exists that utilizes few-shot intrusion generation to assist with few-shot attack classification, especially showing the classification results for the few-shot attacks that are hard to obtain in reality. Meanwhile, a suitable explanation of the training process for the proposed method is demanding. As a result, MAGET is proposed to cover this gap.

3 Proposed method

We describe the threat model and system design of the proposed MAGET. Meanwhile, all symbols and abbreviations related to MAGET are collected in Table 2 for reference.

3.1 Threat Model

A threat model of public IoT environment that connects Internet is designed, where the NIDSes are deployed to detect the few-shot attacks and other malicious traffic for IoT gateways and servers. An abstract view is illustrated in figure 1, which contains two parts, i.e., the device cluster and the server cluster.

The device cluster includes IoT devices that record physical data from industrial environments and production processes. These devices work as a bridge between physical and virtual space for the whole system. The network packets collected by

Table 2 List of Symbols and Abbreviations

	Description		Description		Description
A	Generated amount	B	Task batch size	C_a	Anomaly-based classifier
C_s	Signature-based classifier	D	Dataset	D_f	Full decoder
Dis	Discriminator	E_f	Full encoder	E_p	Partial encoder
G	Generator	J	Joint function	K	Meta-training task amounts
K'	Meta-testing task amounts	M	Sample, query subset size	M_s	Support subset size
T	Task	x	Original features	x_{func}	Functional features
x_{gen}	Generated features	x_{nonf}	Non-functional features	x_{recon}	Reconstructed features
y_a	Anomaly-based Label	y_s	Signature-based Label	z	Gaussian noise vector
z_0	Zero vector	Φ	Model parameters	Φ'	Initial model parameters
η	Learning rates for searching	ξ	Learning rates for updating	ζ	Loss function

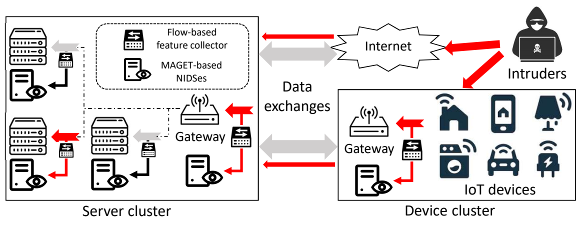


Fig. 1 Overall threat model for public IoT that connects Internet. A device cluster communicates with a server cluster for data exchange. Intruders release malicious packets to devices or gateways. MAGET-based NIDSes are bypassed deployed to inspect them from massive normal traffic.

these devices are converged into a central gateway. A flow-based feature collector is assumed to be deployed at this gateway in a bypass model for recording the global traffic flows.

The server cluster contains the IoT servers that are responsible for providing all sorts of commercial services for devices, such as real-time interaction and data analysis. Another central gateway is included to receive or send traffic with the gateway in the device cluster. This gateway also connects to Internet for web service usage. A set of flow-based feature collectors are assumed at both IoT servers and this central gateway to record the global and local traffic flows.

All these collectors record the network flows with statistical attributes, such as protocol type, flow duration, and the incoming/outcoming number of bytes. An MAGET-based NIDS is linked to every collector to inspect the network flows.

Intruders tend to choose the central gateways and devices as attack entrances. They execute several types of attack traffic under the network layer and application layer, as follows:

DoS/DDoS. They attempt to paralyze the target system with a great number of meaningless traffic from massively compromised machines. The traffic protocols include TCP, UDP, and HTTP.

Probing attacks. It collects information about the target by scanning the whole system. It includes Service Scan and OS Fingerprint.

Information theft. It is supposed to compromise the target server and steal sensitive credentials or unauthorized data on the SSH service. It includes Keylogging and Data Exfiltration.

Web attack. It intrudes on the server cluster through the Internet at the application layer. It includes SQL Injection that forces the server to reply information, Cross-Site Scripting (XSS) that injects scripts, and Bruteforce over HTTP to gain access by combinations of username and password.

3.2 System design

MAGET is proposed to generate few-shot attack instances to assist few-shot attack classification. MAGET is both GAN-based and MAML-based, its system design is shown in figure 2. Basically, MAGET has a GAN-based attack generation part and a prediction part that includes both anomaly-based and signature-based classification models. Both parts are trained in a MAML-based way. Concretely, the first part is pre-trained by non-few-shot attacks and fine-tuned by few-shot attacks to achieve attack generation. The second part is pre-trained by non-few-shot traffic and fine-tuned with both original traffic and generated attacks. Two fine-tune steps are supposed to make use of the optimal initial parameters trained by non-few-shot traffic in the pre-trained steps. MAGET also includes a dataset division part based on the rule of MAML and following the work [20][30]. This part converts the original dataset into a multitask form to adapt the training process in a MAML way. Additionally, the

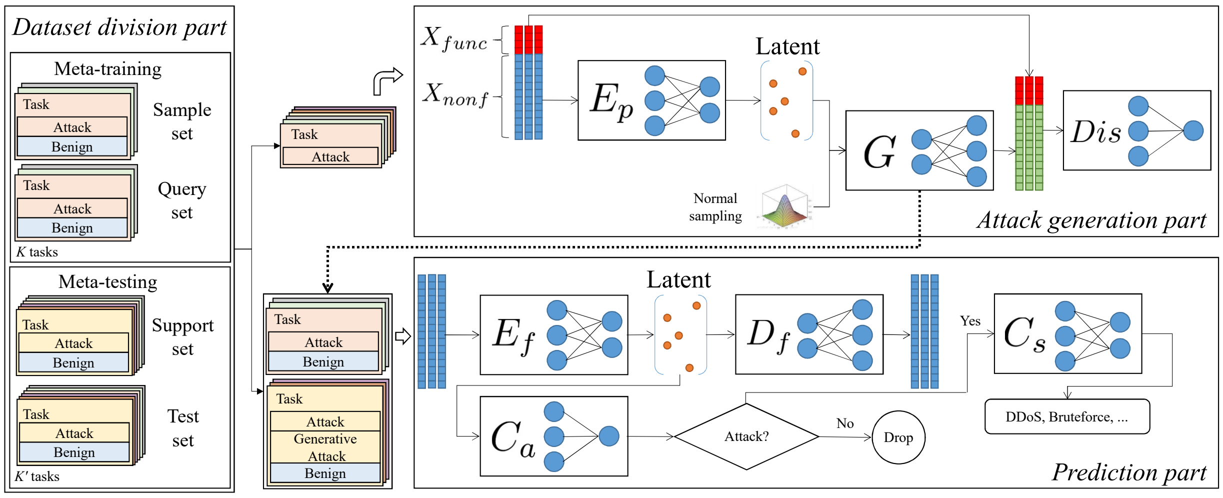


Fig. 2 The system design of MAGET. Data division part divides dataset into four subsets for meta-learning-based usage. Attack generation part applies attack instances to achieve data augmentation based on a hybrid structure of an auto encoder (E_p, G) and GAN (G, Dis). Prediction part applies original and generated instances with another auto encoder (E_f, D_f) to extract features, which are then classified by an anomaly-based classifier (C_a) and signature-based classifier (C_s).

attack generation part can only modify the non-functional features of an attack vector, which do not reflect the functionality of this vector [31]. Meanwhile, auto-encoders are used in both attack generation and prediction parts for stable training and mode-collapse prevention [32].

Dataset division part. All samples are divided into four subsets, which are the sample, query, support and test sets. Each set contains benign instances and instances of one attack type. The sample and query sets are combined into a meta-training set. This set contains attack types that have sufficient samples and is used to search and verify the optimal initial model parameters Φ' for models. The query and test sets are combined into a meta-testing set. This set contains all attack types with a few samples to fine-tune and test models. Indeed, only the support set contains the training samples from few-shot attacks. As for subset size, the meta-training set has K tasks, and each task has M samples. The meta-testing set has K' tasks, K' equals the number of total attack types. Each task in the support set possesses M_s samples. Based on the dataset division, only the attack samples are selected to generate attack samples in the attack generation part, whereas the generative samples, original attacks and benign instances are mixed and put into the prediction part.

Attack generation part. Initially, the features of an original attack vector are divided into functional features x_{func} and non-functional features x_{nonf}. Second, x_{nonf} are fed into an encoder E_p, which then outputs a low-dimensional latent vector. Third, this vector is combined with another noise vector z sampled from the normal distribution. The combined vector is then fed into a generator G. Fourth, the output features from G are connected with x_{func}, which are then treated as a generated attack instance x_{gen}. Overall, the formulation of x_{gen} can be expressed as Equation (3), where J is a joint function. A discriminator Dis is also trained to examine the reality of x_{gen}.

$$x_{gen} = J(x_{func}, G(E_p(x_{nonf}), z)) \tag{3}$$

Prediction part. Initially, the features are converted into latent vectors by an encoder E_f. This encoder is different from E_p because it requires full features rather than only non-functional features. Correspondingly, a decoder D_f is required as an auxiliary model to train E_f by minimizing the reconstruction loss. The latent vectors from E_f are then put into an anomaly-based classifier C_a. If C_a classifies one vector as normal, this vector will be discarded. Otherwise, this vector is put into a signature-based classifier C_s for sub-category classification. The performance

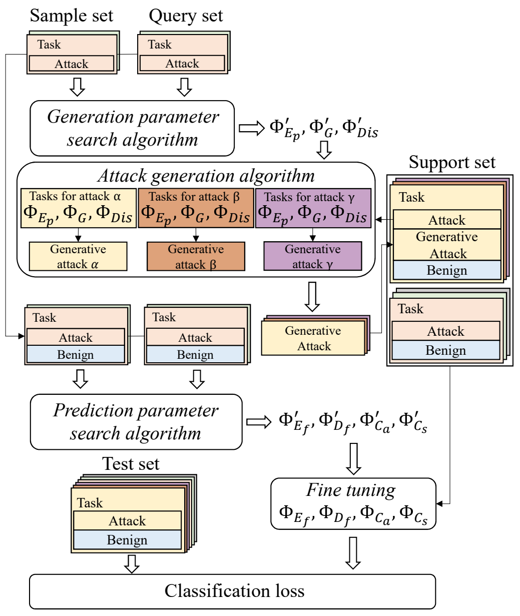

Sample set
Query set
Task
Attack
Generation parameter search algorithm
$\Phi'_{E_p}, \Phi'_G, \Phi'_{Dis}$
Attack generation algorithm
Tasks for attack α
Tasks for attack β
Tasks for attack γ
$\Phi_{E_p}, \Phi_G, \Phi_{Dis}$
Generative attack α
Generative attack β
Generative attack γ
Support set
Generative Attack
Benign
Prediction parameter search algorithm
$\Phi'_{E_f}, \Phi'_{D_f}, \Phi'_{C_a}, \Phi'_{C_s}$
Test set
Fine tuning
$\Phi_{E_f}, \Phi_{D_f}, \Phi_{C_a}, \Phi_{C_s}$
Classification loss

Fig. 3 MAGET training progress. First, sample and query sets are used to find the optimal initialization parameters for E_p, G, Dis. Second, they are fine-tuned by the support set and used for attack generation. Third, the sample and query sets are also used to find the optimal initialization parameters for E_f, D_f, C_a, C_s. Fourth, they are fine-tuned by the support set and generated attack. Finally, test set is utilized to calculate the classification loss.

of C_s is the research focus in MAGET since it reflects the real classification situation for every few-shot attack, whereas C_a treats all few-shot and non-few-shot attacks as anomalies.

4 MAGET training progress and workflow

The training progress of MAGET is illustrated in figure 3. Overall, all models in MAGET are trained by non-few-shot traffic (i.e., the sample and query sets) to get optimal initialization parameters and then fine-tuned by a small amount of traffic that contains few-shot attacks (i.e., the support set). Meanwhile, few-shot attacks in the support set are augmented to improve the capabilities of the prediction models on few-shot attacks.

Concretely, a pre-training algorithm named "generation parameter search algorithm" is first applied to the attack instances in the sample and query sets. This algorithm is responsible for using non-few-shot attack samples to find the optimal initialization parameters for models in the attack generation part (i.e., E_p, G, Dis). This algorithm outputs $\Phi'_{E_p}, \Phi'_G, \Phi'_{Dis}$. Then, "attack generation algorithm" is used to both fine-tune the model parameters $\{\Phi_{E_p}, \Phi_G, \Phi_{Dis}\}$ and implement attack generation based on the instances of few-shot attack types in the support set. All generated attack instances are then utilized to enrich the support set. Next, both benign and attack samples in the sample and query sets are fed into the prediction part, and a new algorithm named "prediction parameter search algorithm" is adopted to look for the suitable initialization model parameters $\{\Phi'_{E_f}, \Phi'_{D_f}, \Phi'_{C_a}, \Phi'_{C_s}\}$. Based on these parameters, one more algorithm "prediction fine-tuning algorithm" is utilized to fine-tune the models with all the samples in the enriched support set, including the generative attacks, original attacks and benign instances. Finally, the test set measures the fine-tuned models based on the classification losses of C_a and C_s.

After the training, E_f, C_a, C_s constructs MAGET-based NIDS. Its workflow is illustrated in figure 4. Hidden in massive normal traffic packets, the malicious packets launched by the intruders are first filtered by a firewall based on some defined rules. Next, the features of these packets are collected by the bypassed collector, which are then processed by the MAGET-based NIDS: E_f extracts the features; C_a identifies those malicious packets as anomalies and raises an alarm, whereas normal packets are dropped; C_s classifies these anomalies into sub-categories. Then, an extra analysis module is deployed to sense the nature of the anomalies, including their goal, means and calamity degree. Based on all the obtained and analyzed information, a mitigation module is responsible for choosing a set of appropriate options to mitigate the impact of these malicious packets. These options include dropping these malicious packets, blocking traffic from the source address, resetting the connections, and reconfiguring the rules in the firewall to prevent future attacks.

The details of the four contained algorithms in MAGET training progress are described below.

4.1 Generation parameter search algorithm

This algorithm is designed to gain optimized Φ'_G, Φ'_{Dis}, Φ'_{E_p} with the attack instances in the sample and query sets. Thus, this algorithm requires the losses of E_p, G and Dis. Considering that E_p and

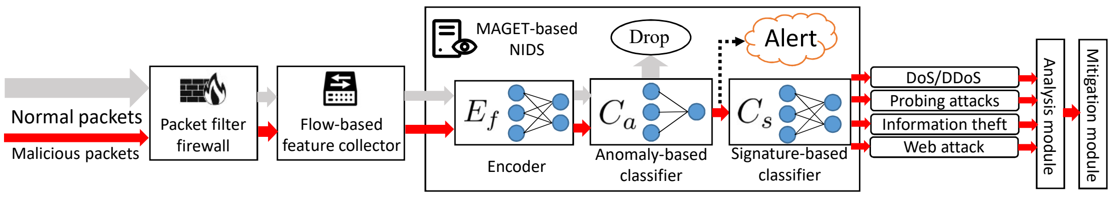


Fig. 4 MAGET-based NIDS workflow. Incoming packets are first filtered by the firewall and then collected by the feature collector. The NIDS first extracts these features by E_f, and then identifies the malicious packets from massive normal packets by C_a. C_s classifies the malicious packets into concrete sub-categories. Next, an analysis module tries to deduce their goals, means and calamity degrees. Finally, a mitigation module takes measures to mitigate the impact of these packets.

G play the function of an auto-encoder, a reconstructed attack sample x_{recon} is calculated by Equation 4. Hence, a mean-squared loss is defined at Equation (5), where M is the amount of the sample and query sets. In addition, x_{recon} is different from x_{gen} because it utilizes a fixed zero vector z_0 instead of a Gaussian noise vector.

$$x_{recon} = J(x_{func}, G(E_p(x_{nonf}), z_0)) \tag{4}$$

$$\zeta_{E_p} = \frac{1}{M}\sum_{i=1}^{M}(x_{recon}^{(i)} - x^{(i)})^2 \tag{5}$$

Then a structure of GAN loss is appended on G and Dis. In addition to minimizing the mean-squared loss, G also generates x_{gen} to bypass Dis. Thus, the loss of G is defined as follow:

$$\zeta_G = \frac{1}{M}\sum_{i=1}^{M}\{(x_{recon}^{(i)} - x^{(i)})^2 - log[Dis(x_{gen}^{(i)})]\} \tag{6}$$

Dis is supposed to distinguish x_{gen} and x. Concretely, its loss function is illustrated in Equation (7):

$$\zeta_{Dis} = -\frac{1}{M}\sum_{i=1}^{M}\{log[1 - Dis(x_{gen}^{(i)})] + log[Dis(x^{(i)})]\} \tag{7}$$

The pseudo-code of this algorithm is listed in Algorithm 1. First, Φ'_G, Φ'_{Dis} and Φ'_{E_p} are assigned with random values. Second, a double loop structure is utilized, where B tasks are sampled as a task batch in its outer loop while each task is traversed in its inner loop. For every task in the inner loop (line $4-12$), ζ_{E_p}, ζ_G and ζ_{Dis} are calculated with the sample set. Then these losses are used to figure out Φ_G, Φ_{Dis} and Φ_{E_p} based on a one-step gradient descent from Φ'. The learning rates in this loop are denoted as $\{\eta_G, \eta_{Dis}, \eta_{E_p}\}$. For every task batch in the outer loop (line $13-18$), ζ_{E_p}, ζ_G and ζ_{Dis} are calculated with the query set. Φ'_G, Φ'_{Dis} and Φ'_{E_p} are optimized with average losses based on B sets of model parameters $\{\Phi_G$, Φ_{Dis}, $\Phi_{E_p}\}$. The learning rates in this loop are denoted as $\{\xi_G, \xi_{Dis}, \xi_{E_p}\}$. In summary, this is a two-step updating process for Φ'. The sample set promotes Φ' to take a step initially, then the query set helps Φ' to find the second-step direction, and Φ' is indeed updated on this direction. Additionally, η is generally set larger than ξ because η is utilized to search the second-step direction while ξ is utilized to update the model parameters. Larger η can find more potential directions as references, whereas smaller ξ makes stable training progress.

4.2 Attack generation algorithm

This algorithm fine-tunes Φ' and generates instances for each few-shot attack type according to the attack instances in the support set. The amount generated for each type should be larger than A, which is defined beforehand.

The pseudo-code of this algorithm is displayed in Algorithm 2. In this algorithm, the amount of the support set is denoted as M_s, which is different for each task. M_s is treated as M in the application of Equations (5), (6), (7). This algorithm first trains E_p, G and Dis with the mean-squared loss and the generative adversarial loss at line $2-11$. The learning rates in the former are $\{\xi_{E_p}, \xi_G, \xi_{Dis}\}$. Then the trained models are utilized to generate instances at line $12-23$ until the generated amount reaches A. Dis is applied to constantly judge the generated instances and

Algorithm 1 Generation parameter search algorithm

```
Require: the attack set, subset size $M$, task batch size $B$, learning rates $\eta_G, \eta_{Dis}, \eta_{E_p}, \xi_G, \xi_{Dis}, \xi_{E_p}$;
Output: optimized initialization parameters $\Phi'_{E_p}, \Phi'_G, \Phi'_{Dis}$

 1: Randomly initialize $\Phi'_{E_p}, \Phi'_G, \Phi'_{Dis}$
 2: while not done do
 3:     Sample batch of tasks $\{T\}_B$ from the attack set
 4:     for every task $T_i$ in $\{T\}_B$ do
 5:         $\Phi_{E_p}, \Phi_G, \Phi_{Dis} \leftarrow \Phi'_{E_p}, \Phi'_G, \Phi'_{Dis}$
 6:         Obtain $X = \{x^{(1)}, ..., x^{(M)}\}$ from the sample set
 7:         Get $X_{gen}, X_{recon}$ by Equation (3), (4), get $\zeta_{E_p}, \zeta_G, \zeta_{Dis}$ by Equation (5), (6), (7)
 8:         $\Phi^{(i)}_{E_p} \leftarrow \Phi_{E_p} - \eta_{E_p} \frac{d\zeta_{E_p}(\Phi_{E_p})}{d\Phi_{E_p}}$, $\Phi^{(i)}_G \leftarrow \Phi_G - \eta_G \frac{d\zeta_G(\Phi_G)}{d\Phi_G}$, $\Phi^{(i)}_{Dis} \leftarrow \Phi_{Dis} - \eta_{Dis} \frac{d\zeta_{Dis}(\Phi_{Dis})}{d\Phi_{Dis}}$
 9:     end for
10:     Obtain $X = \{x^{(1)}, ..., x^{(M)}\}$ from the query set
11:     Get $X_{gen}, X_{recon}$ by Equation (3), (4), get $\zeta_{E_p}, \zeta_G, \zeta_{Dis}$ by Equation (5), (6), (7)
12:     Update $\Phi'_{E_p}$ by $\Phi'_{E_p} - \xi_{E_p} \frac{1}{B} \sum_{i=0}^{B} \frac{d\zeta_{E_p}(\Phi^{(i)}_{E_p})}{d\Phi'_{E_p}}$
13:     Update $\Phi'_G$ by $\Phi'_G - \xi_G \frac{1}{B} \sum_{i=0}^{B} \frac{d\zeta_G(\Phi^{(i)}_G)}{d\Phi'_G}$
14:     Update $\Phi'_{Dis}$ by $\Phi'_{Dis} - \xi_{Dis} \frac{1}{B} \sum_{i=0}^{B} \frac{d\zeta_{Dis}(\Phi^{(i)}_{Dis})}{d\Phi'_{Dis}}$
15: end while
```

drop the unqualified ones. Finally, these generated instances are appended to the support set for the prediction part.

4.3 Prediction parameter search algorithm

This algorithm is designed to gain optimized Φ'_{E_f}, Φ'_{D_f}, Φ'_{C_a} and Φ'_{C_s} with both attack and benign instances in the sample and query sets. Thus, this algorithm requires the losses of E_f, D_f, C_a and C_s. Considering that E_f and D_f not only cope with non-functional features but all features, a new mean-squared loss ζ_{recon} is defined in Equation (8) for both two models.

$$\zeta_{recon} = \frac{1}{M} \sum_{i=1}^{M} [D_f(E_f(x^{(i)})) - x^{(i)}]^2 \qquad (8)$$

For C_a and C_s, their loss functions are shown in Equations (9)(10) with a binary cross-entropy loss and a cross-entropy loss. In Equation (9), $y_a \in \{0, 1\}$, and "0" and "1" represent "benign" and "anomaly", respectively. In Equation (10), y_s refers to the corresponding attack label integer and $1 \le y_s \le K'$.

$$\zeta_{C_a} = \frac{1}{M} \sum_{i=1}^{M} \{y_a * log[C_a(E_f(x^{(i)}))] + (1 - y_a) * log[1 - C_a(E_f(x^{(i)}))]\} \qquad (9)$$

$$\zeta_{C_s} = -\frac{1}{M} \sum_{i=1}^{M} \{y_s * log[C_s(E_f(x^{(i)}))]\} \qquad (10)$$

The pseudo-code of this algorithm is shown in Algorithm 3. After randomly initializing Φ'_{E_f}, Φ'_{D_f}, Φ'_{C_a} and Φ'_{C_s}, one outer loop and two inner loops are called. For each task in the first inner loop (line $4 - 11$), ζ_{recon} and ζ_{C_a} are calculated with the sample set, which motivates Φ_{E_f}, Φ_{D_f} and Φ_{C_a} to achieve one-step gradient descent. For each task in the second inner loop (line $12 - 17$), ζ_{C_s} is calculated with the sample set, which promotes Φ_{C_s} to update gradient. For each task batch in the outer loop (line $18 - 24$), ζ_{recon}, ζ_{C_a} and ζ_{C_s} are calculated with the query set. Finally, Φ'_{E_f}, Φ'_{D_f}, Φ'_{C_a} and Φ'_{C_s} are optimized with average losses based on B sets of model parameters $\{\Phi_{E_f}$,

Algorithm 2 Attack generation algorithm

Require: the attack set, subset size M_s, Φ'_{E_p}, Φ'_G, Φ'_{Dis}, A, learning rates $\xi_G, \xi_{Dis}, \xi_{E_p}$;
Output: generated attack set D

```
1: $\Phi_{E_p}, \Phi_G, \Phi_{Dis} \leftarrow \Phi'_{E_p}, \Phi'_G, \Phi'_{Dis}$
2: while not done do
3:     for every task $T_i$ in the attack set do
4:         Obtain $X = \{x^{(1)}, ..., x^{(M_s)}\}$ from the support set
5:         Get $X_{gen}, X_{recon}$ by Equation (3), (4), get $\zeta_{E_p}, \zeta_G, \zeta_{Dis}$ by Equation (5), (6), (7)
6:         Update $\Phi_{E_p}$, $\Phi_G$, $\Phi_{Dis}$ by $\Phi_{E_p} - \xi_{E_p}\frac{d\zeta_{E_p}(\Phi_{E_p})}{d\Phi_{E_p}}$, $\Phi_G - \xi_G\frac{d\zeta_G(\Phi_G)}{d\Phi_G}$, $\Phi_{Dis} - \xi_{Dis}\frac{d\zeta_{Dis}(\Phi_{Dis})}{d\Phi_{Dis}}$
7:     end for
8: end while
9: Define $D$ as an empty dataset
10: for every task $T_i$ in the attack set do
11:     Obtain $X = \{x^{(1)}, ..., x^{(M_s)}\}$ from the support set
12:     Get $X^{gen}$ by Equation (3)
13:     for every sample $x$ in $X^{gen}$ do
14:         if $Dis$ judge $x$ as real and $D$.length $< A$ then
15:             $D.append(x)$
16:         end if
17:     end for
18:     if $D$.length $\geq A$ then
19:         break
20:     end if
21: end for
```

Φ_{D_f}, Φ_{C_a}, $\Phi_{C_s}\}$. Same as Algorithm 1, the learning rates in the inner loop are $\{\eta_{E_f}$, η_{D_f}, η_{C_a}, $\eta_{C_s}\}$, while the ones in the outer loop are $\{\xi_{E_f}$, ξ_{D_f}, ξ_{C_a}, $\xi_{C_s}\}$.

4.4 Prediction fine-tuning algorithm

this algorithm fine-tunes Φ_{E_f}, Φ_{D_f}, Φ_{C_a} and Φ_{C_s} based on Φ' of Algorithm 3 with the appended support set, whose size is denoted as M'_s. M'_s equals to $M_s + A$ if the attack in the task is few-shot. Otherwise, M'_s equals to M_s.

The fine-tuning process is illustrated in Algorithm 4. This process continuously trains E_f, D_f, C_a and C_s with the mean-squared loss, the binary cross entropy loss and the cross entropy. The learning rates are $\{\xi_{E_f}, \xi_{D_f}, \xi_{C_a}, \xi_{C_s}\}$. After the fine-tuning, Φ_{E_f}, Φ_{C_a} and Φ_{C_s} as a whole can achieve both anomaly-based and signature-based classification. The few-shot attacks are supposed to be identified by Φ_{C_s}.

5 Experiments

This section introduces the experimental results of MAGET. First, the setup and the pre-processing steps of the experiment are introduced. Second, the model training processes are explained, including the loss variation of each model and the generation quality variation of G. Next, the experimental results for anomaly-based and signature-based classification of MAGET are listed and evaluated, while the latter is critically compared with advanced methods. Then, the effectiveness and the efficiency of the few-shot generation are discussed. Finally, the limitations of MAGET are given.

5.1 Experimental setup

Datasets. Two public datasets have been applied, CSE-CIC-IDS2018 [33] from Canadian Institute for Cybersecurity and Bot-IoT [34] from Cyber Range Lab of UNSW Canberra.

CSE-CIC-IDS2018 contains abstract representations of events and behaviors on the network

Algorithm 3 Prediction parameter search algorithm

```
Require: the mixed set, subset size $M$, task batch size $B$, learning rates $\eta_{E_f}, \eta_{D_f}, \eta_{C_a}, \eta_{C_s}, \xi_{E_f}, \xi_{D_f}, \xi_{C_a}, \xi_{C_s}$;
Output: optimized initialized $\Phi'_{E_f}, \Phi'_{D_f}, \Phi'_{C_a}, \Phi'_{C_s}$

1: Randomly initialize $\Phi'_{E_f}, \Phi'_{D_f}, \Phi'_{C_a}, \Phi'_{C_s}$
2: while not done do
3:     Sample batch of tasks $\{T\}_B$ from the mixed set
4:     for each task $T_i$ in $\{T\}_B$ do
5:         $\Phi_{E_f}, \Phi_{D_f}, \Phi_{C_a}, \Phi_{C_s} \leftarrow \Phi'_{E_f}, \Phi'_{D_f}, \Phi'_{C_a}, \Phi'_{C_s}$
6:         Obtain $X = \{x^{(1)}, ..., x^{(M)}\}$ from the sample set, $Y_a = \{y_a^{(1)}, ..., y_a^{(M)}\} \quad s.t. \quad y_a \in \{0, 1\}$
7:         Get $\zeta_{recon}$, $\zeta_{C_a}$ by Equations (8), (9)
8:         $\Phi^{(i)}_{E_f,D_f} \leftarrow \Phi_{E_f,D_f} - \eta_{E_f,D_f} \frac{d\zeta_{recon}(\Phi_{E_f,D_f})}{d(\Phi_{E_f}+\Phi_{D_f})}$, $\Phi^{(i)}_{C_a} \leftarrow \Phi_{C_a} - \eta_{C_a} \frac{d\zeta_{C_a}(\Phi_{C_a})}{d\Phi_{C_a}}$
9:     end for
10:    for every task $T_i$ in $\{T\}_B$ do
11:        Obtain $X = \{x^{(1)}, ..., x^{(M)}\}$ from the sample set, $Y_s = \{y_s^{(1)}, ..., y_s^{(M)}\} \quad s.t. \quad 1 \leq y_s \leq K'$
12:        Get $\zeta_{C_s}$ by Equation (10), $\Phi^{(i)}_{C_s} \leftarrow \Phi_{C_s} - \eta_{C_s} \frac{d\zeta_{C_s}(\Phi_{C_s})}{d\Phi_{C_s}}$
13:    end for
14:    Obtain $X = \{x^{(1)}, ..., x^{(M)}\}$ from the query set
15:    Obtain $Y_a = \{y_a^{(1)}, ..., y_a^{(M)}\} \quad s.t. \quad y_a \in \{0, 1\}$, $Y_s = \{y_s^{(1)}, ..., y_s^{(M)}\} \quad s.t. \quad 0 \leq y_s \leq K'$
16:    Get $\zeta_{recon}$, $\zeta_{C_a}$, $\zeta_{C_s}$ by Equation (8), (9), (10)
17:    Update $\Phi'_{E_f,D_f}$, $\Phi'_{C_a}$, $\Phi'_{C_s}$ by
           $\Phi'_{E_f,D_f} - \xi_{E_f,D_f} \frac{1}{B} \sum_{i=0}^{B} \frac{d\zeta_{recon}(\Phi^{(i)}_{E_f,D_f})}{d(\Phi'_{E_f}+\Phi'_{D_f})}$, $\Phi'_{C_a} - \xi_{C_a} \frac{1}{B} \sum_{i=0}^{B} \frac{d\zeta_{C_a}(\Phi^{(i)}_{C_a})}{d\Phi'_{C_a}}$, $\Phi'_{C_s} - \xi_{C_s} \frac{1}{B} \sum_{i=0}^{B} \frac{d\zeta_{C_s}(\Phi^{(i)}_{C_s})}{d\Phi'_{C_s}}$
18: end while
```

Algorithm 4 Prediction fine-tuning algorithm

```
Require: the mixed set, subset size $M'_s$, $\Phi'_{E_f}$, $\Phi'_{D_f}$, $\Phi'_{C_a}$, $\Phi'_{C_s}$, learning rates $\xi_{E_f}, \xi_{D_f}, \xi_{C_a}, \xi_{C_s}$;
Output: Final detection model set $\{\Phi_{E_f}, \Phi_{C_a}, \Phi_{C_s}\}$;

   $\Phi_{E_f}, \Phi_{D_f}, \Phi_{C_a}, \Phi_{C_s} \leftarrow \Phi'_{E_f}, \Phi'_{D_f}, \Phi'_{C_a}, \Phi'_{C_s}$
1: while not done do
2:     for every task $T_i$ in the mixed set do
3:         Obtain $X = \{x^{(1)}, ..., x^{(M'_s)}\}$ from the support set
4:         Obtain $Y_a = \{y_a^{(1)}, ..., y_a^{(M'_s)}\} \quad s.t. \quad y_a \in \{0, 1\}$, $Y_s = \{y_s^{(1)}, ..., y_s^{(M'_s)}\} \quad s.t. \quad 1 \leq y_s \leq K'$
5:         Get $\zeta_{recon}$, $\zeta_{C_a}$, $\zeta_{C_s}$ by Equation (8), (9), (10)
6:         Update $\Phi_{E_f,D_f}$, $\Phi_{C_a}$, $\Phi_{C_s}$ by
               $\Phi_{E_f,D_f} - \xi_{E_f,D_f} \frac{d\zeta_{recon}(\Phi_{E_f,D_f})}{d(\Phi_{E_f}+\Phi_{D_f})}$, $\Phi_{C_a} - \xi_{C_a} \frac{d\zeta_{C_a}(\Phi_{C_a})}{d\Phi_{C_a}}$, $\Phi_{C_s} - \xi_{C_s} \frac{d\zeta_{C_s}(\Phi_{C_s})}{d\Phi_{C_s}}$
7:     end for
8: end while
```

including seven main network attack scenarios (i.e., Bruteforce, Heartbleed, Botnet, DoS, DDoS, Web attacks and Infiltration). To simulate and collect these attack instances, 50 nodes of attack infrastructures and 420 hosts of victim organizations have been deployed. All collected network traffics are converted into 80-feature vectors by CICFlowMeter [35].

Table 3 Overall Characteristics of CSE-CIC-IDS2018 and Bot-IoT

Dataset	CSE-CIC-IDS2018 (CCI)	Bot-IoT (BI)
Release Year	2018	2019
Platform/Tool	Windows and Linux workstations, CICFlowMeter	VMs of Windows, Ubuntu and Kali, Node-red, Argus
Feature amount	79 (except “Label”)	29 (46 in 5% subset)
Features	Destination Port, Idle Std, Bwd Packet Length Std, ...	eflgs, proto, state, pkts, bytes, dur, spkts, dpkts, ...
Numbered Labels	0(Benign), 1(Bot), 2(DDoS LOIC HTTP), 3(FTP Bruteforce), 4(SSH Bruteforce), 5(DoS GoldenEye), 6(DoS Slowloris), 7(DoS Hulk), 8(DoS SlowHTTPTest), 9(DDoS HOIC), 10(Infiltration), 11(DDoS LOIC UDP), 12(Bruteforce Web), 13(Bruteforce XSS), 14(SQL Injection)	0(Normal), 1(DoS TCP), 2(DoS UDP), 3(DDoS TCP), 4(DDoS UDP), 5(Service Scan), 6(OS Fingerprint), 7(DoS HTTP), 8(DDoS HTTP), 9(Keylogging), 10(Data Exfiltration)

Bot-IoT is a state-of-the-art dataset for profiling the combination of normal and botnet under an IoT-specific network. Node-red [36] was utilized to simulate IoT services. Concretely, the network traffic was collected under a testbed including multiple Virtual Machines (VMs) with firewalls and taps, which was then transferred into CSV form by Argus [37]. Additionally, the authors also provided a 5% subset as a smaller and more manageable version. The details of these datasets are listed in Table 3. For convenience, CCI and BI are used for short in the following tables and figures.

Evaluation metrics. Based on True Negative (TN), True Positive (TP), False Negative (FN) and False Positive (FP), several mathematical metrics are adopted for both anomaly-based and signature-based models, including Accuracy (Acc), True Positive Rate (TPR), False Positive Rate (FPR), Precision (Pre), Recall (Rec) and F1-score. Acc measures the model performance in the most common way. For the anomaly-based model, TPR reflects the ratio of correctly detected intrusions, and FPR shows the ratio of the normal samples that are incorrectly classified as intrusions. For the signature-based model, Pre quantifies the number of correct intrusions predicted by the model, Rec plays the same role as TPR, and F1-score judges the model by the average of these two indicators.

Besides, Receiver Operating Characteristic (ROC) curve is used to visualize the relationship between TPR and FPR for the anomaly-based model with different output thresholds. Area Under Curve (AUC), the entire area underneath the ROC curve, is used to measure aggregate performance across all these thresholds. A confusion matrix is used to show the classification details of the signature-based model.

As for the performance of G, Fréchet Inception Distance (FID) is calculated, which reflects the quality of generated instances based on the distance between generated and real vectors in feature space. Unlike directly utilizing Inception V3 [38] as a pre-trained feature extractor in the domain of the image classification, the calculation process of the FID in our experiments is well-designed as shown in figure 5. First, the non-functional features of the original attack instances are utilized to train a DNN-based multi-classifier. Second, this classifier deletes its last layer and then works as a feature extractor for all attack instances. Third, FID is calculated based on the latent vectors from this extractor.

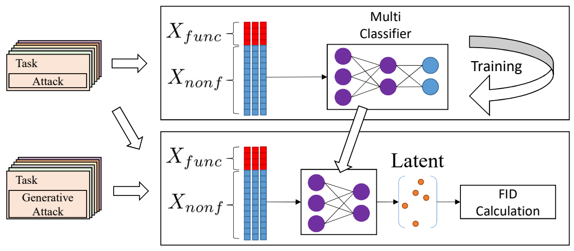


Fig. 5 FID calculating workflow for attack instances. A DNN-based multi-classifier is trained based on non-functional features of attack instances. Then, it is transfered as the pre-trained feature extractor for FID calculating except its last layer.

Hyper-parameters definition. The configuration of hyper-parameters is defined in Table 4. Adam was chosen as the optimizer for each model. The learning rates η and ξ are set to 0.4 and 0.001. All models share the same η and ξ because these models do not differ much on scale. As for subset construction, the sample and query sets are sampled from non-few-shot traffic. These two sets have the same size of $K \times M$. To make use of the few-shot attacks in these two datasets, 70% few-shot attack samples are chosen as training samples, and they are included in the task of the support set. Also, the support set contains 2000 instances to in non-few-shot tasks. The test set contains the final remaining instances in these two datasets. Additionally, the amount of every label in this set is upsampled or downsampled into 1000 for averaging the classification contribution per label.

Comparison setting. SVM, RF, MAML and GAN-enhanced DNN are chosen as baseline methods for comparison. SVM utilizes linear kernel with penalty 1. RF possesses 100 trees with gini criterion. MAML applies the same subsets as the proposed MAGET, but the support set is not appended. GAN-enhanced DNN adopts a DNN to classify the datasets appended by a vanilla GAN. To verify MAGET effectiveness on signature-based classification, generative-based methods including G-IDS [39] and DDPM [40], ML-based methods including SafeML [41], MMM-RF[42], DL-based methods including GRU-GBM [43], CNN-LSTM [44], RideNN-DNFN [45], and DIS-IoT [46] are reproduced as advanced methods for C_s comparison. Considering few meta-learning-based works for signature-based intrusion classification, FC-Net [20] and FSL-Capsule [26] are reproduced with adjustment to achieve multi-classification (the input/output of C-Net and the dimension of Delta Score are adjusted in FC-Net whereas the input of Similarity metric, Sigmoid function and output dimension are adjusted in FSL-Capsule). All methods have been repeated 7 times to obtain the average results with upper and lower bounds.

5.2 Pre-processing steps

After the dataset division, some pre-processing steps are required for the execution of MAGET.

First, all subsets were normalized to alleviate the errors from the big values. Concretely, all features were mapped to corresponding positive numbers less than 1 by Equation (11).

$$x'_{(i)} = \frac{x_{(i)} - x_{min}}{x_{max} + x_{min}} \tag{11}$$

Second, the categorical features in all subsets were converted into binary features by one-hot encoding. Specifically, the features of destination port (i.e., “Dst Port”, “sport” and “dport”) were not processed directly since it possesses thousands of values. Instead, the destination ports with less than 500 training samples were combined as a new binary feature “Dst Port Others”, whereas other destination ports were encoded to their respective binary features.

Third, all features in subsets were divided into functional and non-functional ones for attack generation in the figure 2. This division includes two steps. The first step selects all categorical features as functional features by following the work [47]. One advantage is that the features such as protocol and destination port are directly treated as functional. This ensures compliance with the network protocol format of the generated attacks. The second step follows the work [48], where the authors divided CICIDS2017 by utilizing the analysis from the creator of this dataset [22]. The analysis is conducted by RandomForestRegressor [49] to calculate the importance of each feature for each attack type. Thus, the second step uses this analysis to obtain the top five important numerical features in each attack type, which are then treated as functional features. These features with analyzed weights are recorded in Table A.1, and then combined together. As a result, the functional features are shown in Table 5. It should be noted that the second step is different from the previous works [48, 50], which coped with the attack generation based on the functional features of the current attack type. One reason is that Algorithm 1 finds the optimized Φ'_G based on all attack types rather than one type as the meta knowledge.

5.3 Training process

Loss variations. The loss variations of all seven models on both CSE-CIC-IDS2018 and Bot-IoT are shown in figure A.1. Overall, the optimizer of each model seeks the best Φ that minimizes the defined training loss.

Concretely, training losses of E_p, G, Dis for each batch are illustrated in sub-figures A.1(a), (b), (e), (f), where E_p, G, Dis refer to “Encoder”, “Generator” and “Discriminator” respectively. All losses eventually converge, where Φ_{E_p} reaches 0 and Φ_{Dis} is close to 0.69. The former shows a low reconstructed error, whereas the latter indicates that Dis is unable to classify the generated instances from G.

Meanwhile, training losses of E_f, D_f, C_a, C_s for each batch are illustrated in sub-figures A.1(c), (d), (g), (h). “Encoder/Decoder” refers to the loss of E_f and D_f considering that these two models

Table 4 Hyper-parameters configured for constructing and training MAGET

Hyperparameters	**Values or selections**	**Hyperparameters**	**Values or selections**
Chosen optimizer	Adam	B, A	$10, 1000$
$\eta_{E_p}, \eta_G, \eta_{Dis}$	0.4	$\eta_{E_f}, \eta_{D_f}, \eta_{C_a}, \eta_{C_s}$	0.4
$\xi_{E_p}, \xi_G, \xi_{Dis}$	0.001	$\xi_{E_f}, \xi_{D_f}, \xi_{C_a}, \xi_{C_s}$	0.001
Epochs	50	Layer types	Linear with batch normalization
Output size of E_p, E_f and size of z	16	Activation functions	Sigmoid (last layer of Dis, C_a), Tanh (last layer of G), Relu
Sample, query set size	$16000 (K = 1000, M = 16)$	Test set size	14000 (CCI),10000 (BI)
Support set size	$M_s = 1470$ (DDoS LOIC UDP), $M_s = 519$ (Bruteforce Web), $M_s = 195$ (Bruteforce XSS), $M_s = 73$ (SQL Injection), $M_s = 520$ (DoS HTTP), $M_s = 348$ (DDoS HTTP), $M_s = 30$ (Keylogging), $M_s = 3$, $M_s = 2000$ (Non-few-shot tasks), $K' = 14$ (CCI), 10 (BI)		
Few-shot numbered labels	CCI: 11 (DDoS LOIC UDP), 12 (Bruteforce Web), 13 (Bruteforce XSS), 14 (SQL Injection); BI: 7 (DoS HTTP), 8 (DDoS HTTP), 9 (Keylogging), 10 (Data Exfiltration)		

Table 5 Functional features of CSE-CIC-IDS2018 and Bot-IoT

CCI	Categorical: Protocol, Dst Port Numerical: Flow Pkts/s, Flow IAT Std, Pkt Size Avg, Pkt Len Max, Pkt Len Std, Init Fwd Win Byts, Init Bwd Win Byts, Bwd Seg Size Avg, Bwd Pkts/s, Bwd Header Len, Bwd Pkt Len Mean, Bwd Pkt Len Max, Bwd Pkt Len Std, Fwd Act Data Pkts, Fwd Pkts/s, Fwd Pkt Len Max, Fwd Pkt Len Std, Fwd Seg Size Avg, Fwd Pkt Len Mean, Fwd Seg Size Min, Fwd Header Len, Subflow Bwd Pkts, Subflow Fwd Byts, Tot Bwd Pkts, Tot Fwd Pkts, TotLen Fwd Pkts
BI	Categorical: proto, state, sport, dport Numerical: AR_P_Proto_P_Dport, AR_P_Proto_P_Sport, ltime, sbytes, AR_P_Proto_P_SrcIP, TnP_PDstIP, N_IN_Conn_P_DstIP, N_IN_Conn_P_SrcIP, TnBPDstIP, mean, drate, TnP_Per_Dport, TnP_PerProto, sum, dur, Pkts_P_State_P_Protocol_P_DestIP, stddev

possess the same loss (i.e., the loss of Equation (8)). "Anomaly classifier" and "Signature classifier" refer the losses of C_a and C_s, respectively. During the training, the loss of E_f and D_f reach 0, while the losses of C_a and C_s keep decreasing, and finally stabilizes at low values.

FID variation. FID is utilized to show the generation quality of G. As a result, the FIDs of the generated instances based on few-shot attacks are shown in four sub-figures of figure A.2, where each epoch is recorded during the use of Algorithms 1 and 2. Overall, the FID values of all attack types basically decrease from the beginning, and finally maintain a low value below 100. It indicates that these instances are trained by Algorithms 1 and 2 to obey the data distribution as closely as possible to original attacks. Furthermore, two interesting observations exist in these sub-figures. First, the curves in Algorithm 1 are rougher than most curves in Algorithm 2. One explanation is that Algorithm 1 utilizes multiple attack labels to find the optimal Φ. Compared with Algorithm 2, which only uses a single attack label, the quality of samples generated by Algorithm 1 is more unstable. Second, although most curves in Algorithm 2 are smooth, Keylogging and Data Exfiltration in sub-figure A.2(d) still have obvious fluctuations, one reason is the small training amounts (i.e., 30 and 3).

5.4 Results of anomaly-based classification and evaluation

This part shows the performance of the proposed C_a, whose objective is to classify attacks of all types from the whole traffic.

The average ROC curves of C_a and other baseline methods for two datasets are recorded and depicted in figure A.3, where "Anomaly classifier" refers to C_a. As can be seen from the left sub-figure, compared with other baseline methods, C_a has the best performance at the optimal threshold on the CSE-CIC-IDS2018 dataset. Specifically, it achieves 94.3% TPR and 1.8% FPR at this point. The right sub-figure shows that all methods have nearly perfect ROC curves on the Bot-IoT dataset. In fact, C_a achieves 99.8% TPR and 0.1% FPR at the optimal threshold of its curve. One reason is that the task of anomaly detection on Bot-IoT is relatively easy.

Based on these ROC curves, the AUCs of C_a and other baseline methods are presented in Table 6. For the CSE-CIC-IDS2018 dataset, C_a possesses the highest AUC with a mean of 96.5%. This means that at the optimal threshold in the ROC curve, C_a possesses a higher TPR while maintaining a lower FPR compared to other methods.

Table 6 AUC comparison of C_a with baseline methods (%)

Dataset	SVM	RF	MAML	GAN-enhanced DNN	Proposed C_a
CCI	80.8 ± 2.5	95.0 ± 0.6	93.2 ± 3.8	94.6 ± 0.9	**96.5 ± 0.7**
BI	100 ± 0	99.8 ± 0.1	99.0 ± 0.9	99.9 ± 0.1	99.9 ± 0

Table 7 Acc comparison of C_s with baseline and advanced methods for few-shot attacks under training samples M_s (%)

Few-shot attacks	DDoS LOIC UDP	Bruteforce Web	Bruteforce XSS	SQL Injection	DoS HTTP	DDoS HTTP	Keylogging	Data Exfiltration	CCI overall	BI overall
M_s	1470	519	195	73	520	348	30	3	-	-
SVM[1][4]	100.0 ± 0.0	34.7 ± 2.7	56.8 ± 3.5	0.7 ± 0.4	79.4 ± 0.0	4.7 ± 0.0	7.7 ± 0.0	0.0	48.1	30.6
RF[1][5]	98.9 ± 0.0	93.2 ± 1.1	90.6 ± 1.1	60.9 ± 2.4	94.7 ± 0.6	33.9 ± 1.3	3.9 ± 0.2	0.7 ± 0.0	85.9	33.3
GAN-enhanced DNN[1][9]	99.8 ± 0.2	81.7 ± 0.8	90.8 ± 0.5	94.4 ± 0.9	80.8 ± 1.3	79.1 ± 4.2	97.8 ± 1.8	98.1 ± 0.5	91.7	89.0
MAML[1][14]	100.0 ± 0.0	78.5 ± 0.0	85.7 ± 0.0	77.1 ± 6.0	74.2 ± 4.3	88.5 ± 4.3	75.7 ± 4.3	100.0 ± 0.0	85.3	84.6
G-IDS[2][39]	100.0 ± 0.0	21.1 ± 0.6	43.2 ± 4.0	80.8 ± 5.9	76.4 ± 2.6	62.6 ± 8.2	64.4 ± 3.8	73.1 ± 2.9	61.3	69.1
DDPM[2][40]	100.0 ± 0.0	71.5 ± 4.9	53.9 ± 9.8	48.0 ± 5.9	78.2 ± 15.1	72.6 ± 11.5	79.8 ± 5.4	85.0 ± 5.7	67.5	79.6
FC-Net[3][20]	99.6 ± 0.1	73.2 ± 0.3	74.2 ± 0.4	64.3 ± 0.1	77.4 ± 1.3	86.5 ± 1.0	95.7 ± 0.2	99.2 ± 0.8	77.8	89.7
FSL-Capsule[3][26]	100.0 ± 0.0	91.7 ± 1.4	89.6 ± 0.7	90.5 ± 5.4	81.3 ± 1.2	84.3 ± 0.5	95.8 ± 0.7	100.0 ± 0.0	93.0	90.4
SafeML[4][41]	98.7 ± 1.3	92.6 ± 1.4	90.1 ± 0.4	56.9 ± 4.1	93.2 ± 0.8	41.4 ± 3.7	32.2 ± 8.9	10.2 ± 9.9	84.6	44.3
MMM-RF[4][42]	100.0 ± 0.0	85.7 ± 3.2	78.3 ± 2.9	63.4 ± 4.4	77.7 ± 0.1	79.2 ± 0.8	45.7 ± 7.3	38.9 ± 11.7	81.9	60.4
GRU-GBM[5][43]	63.0 ± 2.0	44.5 ± 2.3	12.4 ± 6.1	0.0 ± 0.0	59.4 ± 6.8	43.3 ± 3.7	0.0 ± 0.0	0.0 ± 0.0	30.0	25.7
CNN-LSTM[5][44]	85.3 ± 0.3	17.4 ± 1.4	38.9 ± 8.6	11.3 ± 7.9	69.4 ± 3.3	57.7 ± 2.1	21.3 ± 5.5	0.0 ± 0.0	38.2	37.1
RideNN-DNFN[5][45]	94.7 ± 1.6	77.5 ± 2.1	60.7 ± 1.3	33.9 ± 8.4	74.4 ± 0.4	73.8 ± 1.5	62.7 ± 2.7	50.6 ± 13.9	66.7	65.4
DIS-IoT[5][46]	99.3 ± 0.7	79.6 ± 4.9	58.1 ± 0.2	46.5 ± 3.3	69.7 ± 2.9	79.2 ± 0.5	54.8 ± 3.6	33.2 ± 12.6	70.9	59.2
Proposed C_s	**100.0 ± 0.0**	**94.1 ± 0.3**	**91.4 ± 0.3**	**95.4 ± 0.4**	78.2 ± 1.9	**89.8 ± 1.8**	**99.4 ± 0.6**	**100.0 ± 0.0**	**95.2**	**91.9**

[1] Baseline methods.
[2] Generative-based advanced methods.
[3] Meta-learning-based advanced methods.
[4] ML-based advanced methods.
[5] DL-based advanced methods.

5.5 Results of signature-based classification and evaluation

This part illustrates the experimental results of the proposed C_s, whose objective is to classify attacks into the correct label.

First, the classification results for the specific few-shot attacks compared with baseline and advanced methods are in Table 7. M_s is the actual training amount of these few-shot attacks, which are used in the support set. Some interesting observations are listed as follows:

- C_s has an average multi-classification Acc of 100% on DDoS LOIC UDP, 94.1% on Bruteforce Web, 91.4% on Bruteforce XSS, 95.4% on SQL Injection, 78.2% on DoS HTTP, 89.8% on DDoS HTTP, 99.4% on Keylogging and 100% on Data Exfiltration. Overall, C_s has the highest averaging Acc on two datasets, which are 95.2% on CSE-CIC-IDS2018 and 91.9% on Bot-IoT. It improves 2.2% and 1.5% Acc compared with the second-highest method [26].
- Meta-learning-based methods also gain good performance, including MAML, FC-Net, and FSL-Capsule. FSL-Capsule get the second-highest results, which are 93% and 90.4% Acc. It shows these methods are good to cope with the few-shot sample issue by meta-knowledge. But there is still a gap compared with C_s due to the usage of data augmentation.
- As a generative-based method, GAN-enhanced DNN gets good results, which are 91.7% and 89.0% Acc. However, the performances of another two generative-based methods, G-IDS and DDPM, are unsatisfactory (i.e., 61.3% $\sim$ 79.6%). Section 1 has mentioned that the generative-based models like GAN require sufficient samples to train. Thus, the insufficient few-shot attack training samples can cause instability of the generative-based models, causing an equivocal effect of data augmentation.
- The ML- and DL-based methods do not perform very well on few-shot attack classification, especially for Bruteforce XSS, SQL injection, Keylogging and Data Exfiltration. Meanwhile, RNN-based methods like GRU-GBM and CNN-LSTM have lower Acc than other methods like RideNN-DNFN and DIS-IoT. One reason is that the hidden states of the recurrent structures are mainly occupied by the non-few-shot

traffic due to its training amount, which interferes with the judgment on the few-shot attacks.

To evaluate the performance of C_s on overall datasets that contain both non-few-shot and few-shot attacks, the results compared with advanced methods on different parts of attack labels (i.e., all labels, few-shot attacks, and non-few-shot attacks) are collected in figure 6 and 7. In two figures, C_s and almost all advanced methods possess near Acc which varies from 82.8% to 89.9% in CSE-CIC-IDS2018 and from 96.4% to 99.7% in Bot-IoT. It implies that all methods have close performance on non-few-shot attacks that have enough training samples. However, C_s has the best average performance on few-shot attacks in both datasets (i.e., 95.2% and 91.9%), which makes C_s gain the highest Acc on overall attacks (i.e., 91.4% and 96.3%). Thus, C_s can concentrate on the few-shot attack classification without lowering the performance on non-few-shot attacks.

In addition, the specific classification of each attack type is recorded through confusion matrices in figure 8 and 9. The values of horizontal and vertical coordinates in these matrices represent the label numbers shown in Table 3. In CSE-CIC-IDS2018, C_s performs well on almost all labels, although it still has some shortcomings on DDoS LOIC HTTP, FTP Bruteforce, and DoS GoldenEye classifications. In Bot-IoT, C_s also has excellent performance on almost all labels, while the results on DoS/DDoS HTTP are acceptable.

5.6 Discussion

The effectiveness of few-shot generation to assist detection. We have appended a supplemental experiment about the impact of different A and M_s to C_s on the few-shot attacks. Its results are collected at Table A.2 and A.3. Data Exfiltration is removed when $M_s = 5, 10, 20$ because its amount is 3. These two tables have two findings. One is that C_s performs better with more generated few-shot instances by G, where C_s gets the best performance 97.3% Acc on CCI and 92.4% Acc on BI when $A = 10000$ with the most training samples. One is that C_s performs better with more original few-shot training samples. In conclusion, G does generate few-shot attacks based that can improve C_s to classify few-shot attacks. This improvement is larger with more original training samples or generated amount.

However, A is not recommended to be too huge since it may lower C_s performance on non-few-shot traffic. We also append a supplemental experiment about the impact of different A to C_s on both few-shot and all attack types. Its results are collected at Table A.4. This table shows that C_s has the highest performance on all attack types, which are 91.4% and 96.3% Acc when $A = 1000$. However, this performance decreases into 89.4% and 94.3% when $A = 10000$. Thus, $A = 1000$ is more suitable to balance the classification on few-shot attacks and all attack types.

This effectiveness is due to the missing data of the original few-shot attacks [39]. The original few-shot attacks have only a few samples, whose data distribution may not be apparent to IDS. MAGET can generate the few-shot attacks through the rational non-functional feature modification. These generated attacks can enrich and slightly extend its data distribution, which helps C_s has better results and a good generalization capability. This generalization capability is also visualized in Section 6.

The efficiency of attack generation. We have attached a supplemental experiment to verify the efficiency of MAGET. As a result, the average training time and FID of G compared with other generative-based methods are recorded in Table A.5 and A.6. These tables show that G in MAGET gets the lowest FIDs, i.e., 23.0 and 4.1, with 51 and 16 seconds in both datasets among the generative-based methods. This result indicates the instances generated by MAGET are more stable with less training time compared with other methods.

This efficiency is due to pre-training G with non-few-shot attacks and fine-tuning it with few-shot ones. The few-shot and non-few-shot attacks share the same traffic forms and have closer statistical characteristics than random features. Thus, the parameters pre-trained by non-few-shot attacks are generally closer to the final parameters for few-shot attacks compared with random parameters. As a result, fine-tuning G from these pre-trained parameters is fast-adaptive and can gain better generation performance than training from random parameters. This fast-adaptive property is also visualized in Section 6.

The combination of anomaly-based and signature-based sub-modules. We have evaluated the MAGET performance of independent C_a, C_s, and their combination. Table A.7 shows Acc

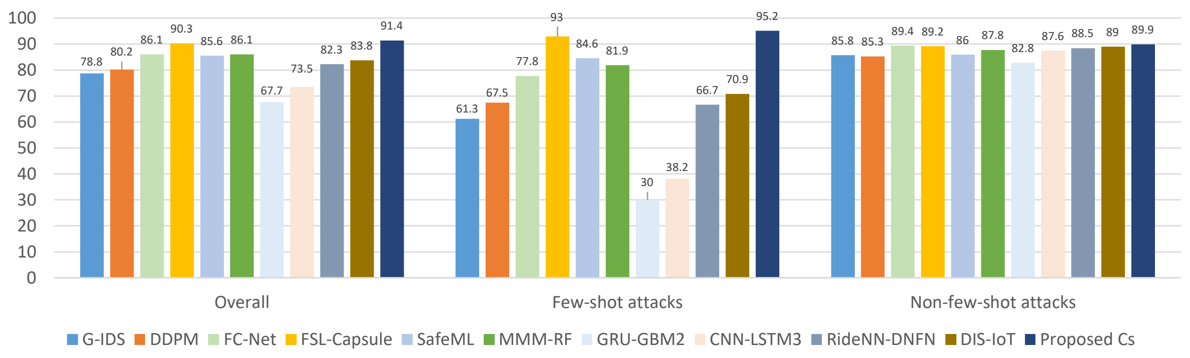


Fig. 6 Acc comparison C_s with advanced methods for different parts of attacks in CSE-CIC-IDS2018 (%)

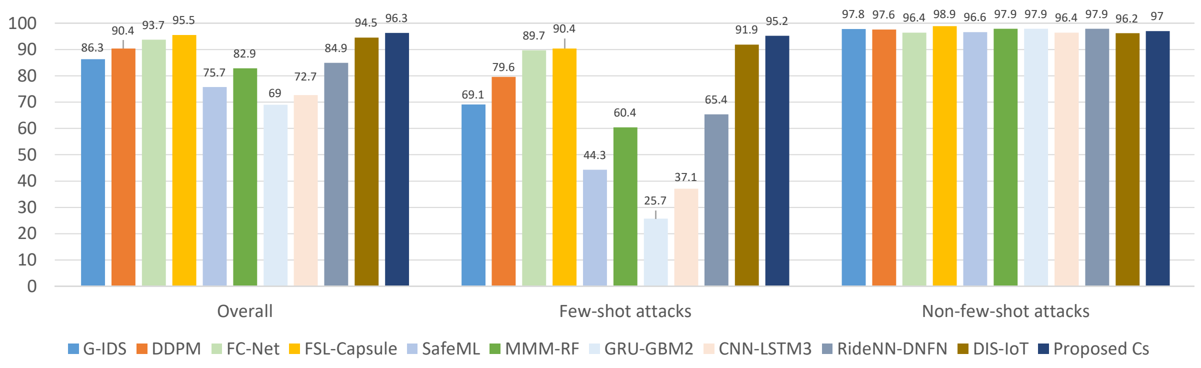


Fig. 7 Acc comparison of C_s with advanced methods for different parts of attacks in Bot-IoT (%)

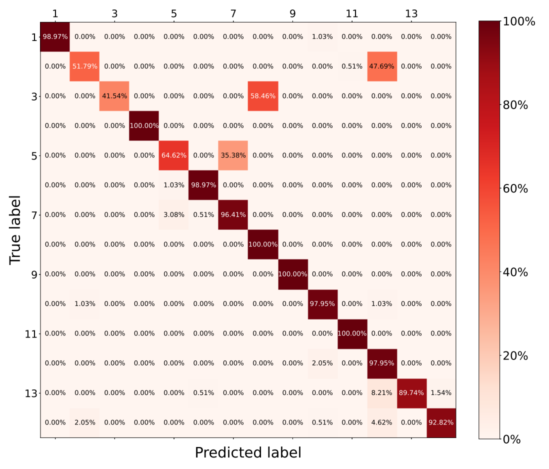


Fig. 8 Confusion matrix of C_s in CSE-CIC-IDS2018

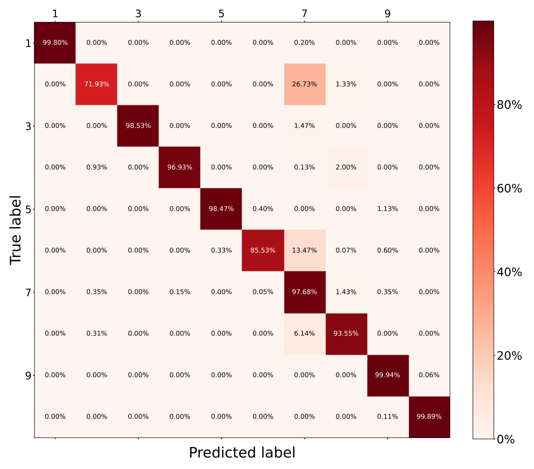


Fig. 9 Confusion matrix of C_s in Bot-IoT

and F1 scores, where the Multi-Stage approach [51] is set as a reference. In two datasets, MAGET gets 94.8% and 99.1% overall F1-scores, respectively. It outperforms the Multi-Stage approach in both sub-modules and combination, due to a significant advantage in its signature-based classification sub-module (i.e., C_s). One reason is that Multi-stage is weak at few-shot detection, while MAGET fills this gap to improve multi-class performance.

Ablation study. We have appended a supplemental Ablation experiment by comparing the MAGET with other three conditions. One is without the pre-training part, which degenerates into the method of GAN-enhanced DNN. One is without the auto-encoders, where E_p, E_f and D_f are removed. One is without the attack generation, which degenerates into the method of MAML. As a result, FID of G, TPR/FPR of C_a and Acc of C_s on few-shot attacks are recorded in Table A.8. On each dataset, the proposed MAGET has the lowest FID on attack generation and the highest Acc on signature-based few-shot attack classification. It also has higher TPR and lower FPR on anomaly-based classification compared with the other conditions. Thus, by combining MAML and GAN, MAGET does outperform these two independent methods, whereas the structure of auto-encoders also provides an improvement on stable attack generation to assist few-shot classification.

5.7 Limitations

Although C_s has excellent classification performance on most few-shot attack labels in both datasets, including DDoS LOIC UDP, Bruteforce Web, Bruteforce XSS, SQL Injection, DDoS HTTP, Keylogging and Data Exfiltration, C_s does have drawbacks to classify DDoS LOIC HTTP, FTP Bruteforce, and DoS GoldenEye in CSE-CIC-IDS2018 and DoS HTTP in Bot-IoT. This is because a single signature-based model rarely achieves a perfect performance for all subcategories. This can be improved by using an architecture of plural models for each label or more careful hyperparameter tuning.

Another limitation is the functional and non-functional feature division. This division strategy chose categorical and numerical features that are important (i.e., differ sharply between attacks and normal traffic) as functional features. However, this strategy still has the possibility of missing some functional numerical features, which are not selected by RandomForestRegressor. Domain expertise will be applied in the future to determine the ultimate set of functional features for newest intrusion datasets.

6 Visualization

In this section, the fast-adaptive property and generalization capability of MAGET are further explained by the variation of model parameters under two-dimensional visualization. The visualization includes the training processes both in the attack generation part and the prediction part.

6.1 Fast-adaptive property on attack generation

The visualization of attack generation is shown in the left part of figure 10, where γ is set as the few-shot attack label whereas α, β, δ are attack labels with enough training samples. In the left sub-figure, $\Phi_{E_p}, \Phi_G, \Phi_D$ are trained to generate attack instances of γ. The closed shapes indicate the suitable parameter ranges that can generate qualified attack instances. Meanwhile, there are local optimum traps, where Φ is almost impossible to escape and converge. The positions of these traps vary depending on the training samples, but all the traps are related to the samples of γ.

We illustrate a comparison between direct training and MAGET-based methods. On the one hand, the direct training method does not draw support from the meta-knowledge of other labels, and its objective is to motivate Φ to move from "Initialized Φ'" to $\Phi^{(4)}$ by the samples of γ. However, this method has two disadvantages. One is that the training process is laborious and unstable due to the small set. The second is that Φ is easy to fall into the local optimal trap, so it is difficult to reach $\Phi^{(4)}$ or its closed shape. Based on the samples of α, β, δ, the MAGET-based method utilizes Algorithm 1 as the meta-training process to find the optimized initial model parameters. As a result, the locations of $\Phi^{(1)}, \Phi^{(2)}, \Phi^{(3)}$ are calculated and their central point Φ' is selected. Since the samples of γ are not applied. This process is not affected by the γ local optimum trap, because it does not apply the samples of γ. Instead, the process may face α, β, δ traps, but even if Φ falls into the trap of one attack type, it still has the ability to jump out of the traps of the other two attack types. It only falls into the traps that α, β, δ have simultaneously, but this rarely happens. Then, Algorithm 2 is applied to motivate Φ to move from Φ' to $\Phi^{(4)}$. Although γ has few samples, the training task is acceptable because the movement is obviously shorter than the direct training method. As a result, a few-shot fast-adaptive process is achieved. This process is less

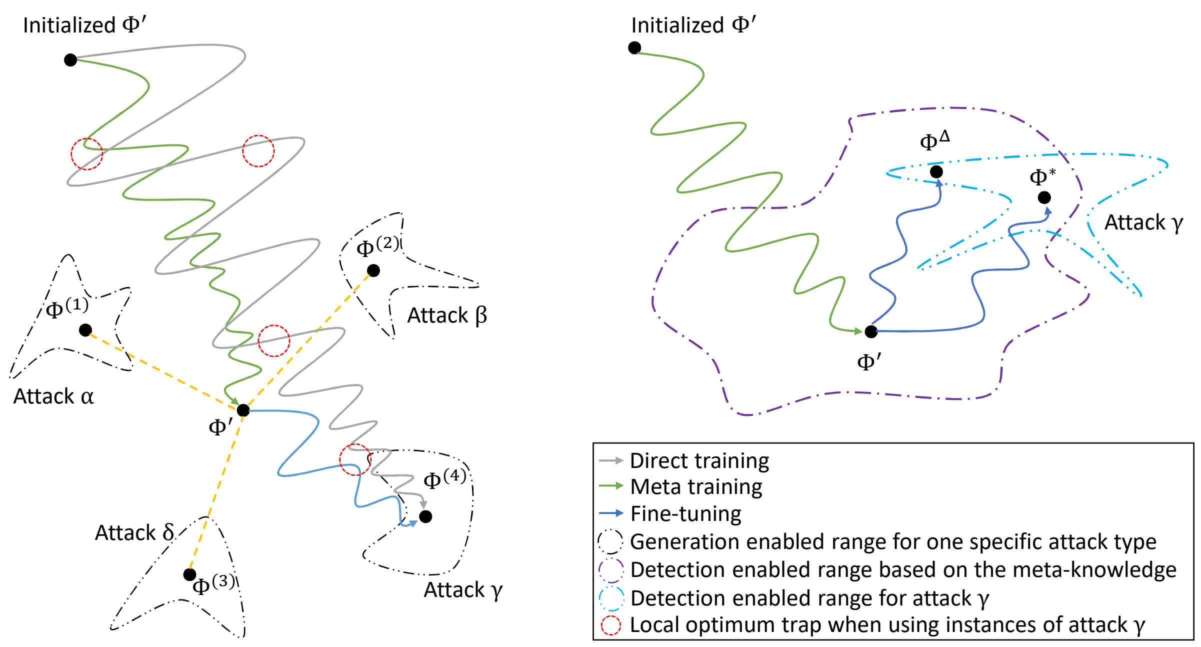

Initialized Φ′
Φ(1)
Attack α
Φ(2)
Attack β
Φ′
Φ(4)
Attack δ
Φ(3)
Attack γ
Initialized Φ′
ΦΔ
Φ*
Attack γ
Φ′
Direct training
Meta training
Fine-tuning
Generation enabled range for one specific attack type
Detection enabled range based on the meta-knowledge
Detection enabled range for attack γ
Local optimum trap when using instances of attack γ

Fig. 10 Two-dimensional Φ visualization in training MAGET. The left part shows the variation of $\Phi_{E_p}, \Phi_G, \Phi_D$ within the attack generation part, whereas the right one depicts the variation of $\Phi_{E_f}, \Phi_{D_f}, \Phi_{C_a}, \Phi_{C_s}$ within the prediction part.

likely to meet the local optimum traps. For example, direct training faces four traps of γ, while MAGET faces only one.

6.2 Generalization capability on prediction

In the right part of figure 10, $\Phi_{E_f}, \Phi_{D_f}, \Phi_{C_a}, \Phi_{C_s}$ are trained to infer incoming traffic, including few-shot attack γ. The closed shapes refer to the parameter range of Φ that enables anomaly-based detection or signature-based classification. The purple closed shape enables models to detect meta-knowledge (i.e.., α, β, δ) whereas the cerulean one concentrates on γ. In the overlap range, Φ can predict all four attack types correctly. Therefore, the objective of the prediction part is to move Φ into this overlap.

We compare MAML and MAGET-based methods. First, both methods search for the optimized initial model parameters Φ' based on the samples of α, β, δ through Algorithm 3. After that, Algorithm 4 is applied to fine-tune the models based on the samples of γ.

In the MAML way, the training samples of γ are sparse and may only reflect a partial data distribution of γ. The fine-tuning step in this way moves Φ' to Φ^{Δ}, which is not optimal for reflecting the real detection range (i.e., the cerulean one in the figure). As a result, the trained models utilize the decision boundaries with a sample-based prejudice to classify attack γ. In other words, the generalization capability of these models on the test set is relatively low.

In the MAGET-based way, samples of γ are extended with instances generated by G. These generated samples have the same functional features as the original ones, whereas the non-functional features are different due to Algorithms 1 and 2. These two algorithms search for a more extensive data distribution to generate these non-functional features (The rationality of this distribution is guaranteed by FID values). Based on the extended samples from this data distribution, the fine-tuning step is able to move Φ' into Φ^* that is better than Φ^{Δ} in the cerulean detection range. As a consequence, this way enables models to have better generalization capabilities.

7 Conclusion and Future Works

In this paper, a Model-Agnostic Generation-Enhanced Technology (MAGET) for few-shot intrusion detection is proposed based on GAN and MAML. It performs few-shot intrusion detection by expanding the samples of few-shot attacks. Based on four learning algorithms, MAGET first transfers meta-knowledge from non-few-shot samples to the model that generates few-shot attacks and then identifies intrusions by using a hybrid detection mechanism. The experiments on CSE-CIC-IDS2018 and Bot-IoT datasets show that MAGET possesses 94.3%/1.8% TPR/FPR and 99.8%/0.1% TPR/FPR in anomaly-based classification and 95.2% and 91.9% accuracy in signature-based classification, respectively. Compared with other related methods, MAGET improves the accuracy of identifying few-shot attacks on these two datasets by at least 2.2% and 1.5%, respectively. Through the analysis of the model parameter visualization process, the models in MAGET are more likely to obtain the global optimum and have better generalization ability than direct training or the original MAML way. In the future, we will implement a real-time testbed with a simulated micro-service topology to analyze traffic from specific applications. Furthermore, we will consider a feature extractor that includes traffic segmentation and sequential embedding to extract network traffic based on the payload and sequential features hidden in adjacent traffic packets.

Acknowledgements. This work was supported by the Opening Project of Intelligent Policing Key Laboratory of Sichuan Province [grant numbers ZNJW2023KFQN00], and was supported by the Chengdu Zhisuan Center for calculating resources.

Declarations

Competing Interests. The authors have no competing interests to declare that are relevant to the content of this article.

Authors contribution statement. Conceptualization: He Junpeng; Methodology: He Junpeng; Formal analysis and investigation: He Junpeng, Yao Lingfeng; Writing - original draft preparation: He Junpeng; Writing - review and editing: Li Xiong, Muhammad Khurram Khan; Funding acquisition: Niu Weina, Zhang Xiaosong; Resources: Li Fagen; Supervision: Li Xiong.

Ethical and informed consent. No data is used for human participants or animals.

Data availability statement. All data generated and analysed during the current study are available from the corresponding author on reasonable request.

References

[1] Muhammad Tariq, Mansoor Ali, Faisal Naeem, and H Vincent Poor. Vulnerability assessment of 6g-enabled smart grid cyber–physical systems. *IEEE internet of things journal*, 8(7):5468–5475, 2020.

[2] Wan Haslina Hassan et al. Current research on internet of things (iot) security: A survey. *Computer networks*, 148:283–294, 2019.

[3] Ansam Khraisat and Ammar Alazab. A critical review of intrusion detection systems in the internet of things: techniques, deployment strategy, validation strategy, attacks, public datasets and challenges. *Cybersecurity*, 4(1):1–27, 2021.

[4] Bhoopesh Singh Bhati and CS Rai. Analysis of support vector machine-based intrusion detection techniques. *Arabian Journal for Science and Engineering*, 45(4):2371–2383, 2020.

[5] Paulo Angelo Alves Resende and André Costa Drummond. A survey of random forest based methods for intrusion detection systems. *ACM Computing Surveys (CSUR)*, 51(3):1–36, 2018.

[6] Anna Drewek-Ossowicka, Mariusz Pietrołaj, and Jacek Rumiński. A survey of neural networks usage for intrusion detection systems. *Journal of Ambient Intelligence and Humanized Computing*, 12(1):497–514, 2021.

[7] TP Latchoumi, Manoj Sahit Reddy, and K Balamurugan. Applied machine learning predictive analytics to sql injection attack detection and prevention. *European Journal of Molecular & Clinical Medicine*, 7(02):2020, 2020.

[8] Yang Guo. A review of machine learning-based zero-day attack detection: Challenges and future directions. *Computer Communications*, 198:175–185, 2023.

[9] JooHwa Lee and KeeHyun Park. Gan-based imbalanced data intrusion detection system. *Personal and Ubiquitous Computing*,

25(1):121–128, 2021.

[10] Junpeng He, Lei Luo, Kun Xiao, Xiyu Fang, and Yun Li. Generate qualified adversarial attacks and foster enhanced models based on generative adversarial networks. *Intelligent Data Analysis*, 26(5):1359–1377, 2022.

[11] Mike Huisman, Jan N Van Rijn, and Aske Plaat. A survey of deep meta-learning. *Artificial Intelligence Review*, 54(6):4483–4541, 2021.

[12] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[13] Tingting Wang, Qiujian Lv, Bo Hu, and Degang Sun. A few-shot class-incremental learning approach for intrusion detection. In *2021 International Conference on Computer Communications and Networks (ICCCN)*, pages 1–8. IEEE, 2021.

[14] Tongtong Feng, Qi Qi, Jingyu Wang, and Jianxin Liao. Few-shot class-adaptive anomaly detection with model-agnostic meta-learning. In *2021 IFIP Networking Conference (IFIP Networking)*, pages 1–9, 2021.

[15] James P Anderson. Computer security threat monitoring and surveillance. *Technical Report, James P. Anderson Company*, 1980.

[16] Zeeshan Ahmad, Adnan Shahid Khan, Cheah Wai Shiang, Johari Abdullah, and Farhan Ahmad. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, 32(1):e4150, 2021.

[17] Nadia Chaabouni, Mohamed Mosbah, Akka Zemmari, Cyrille Sauvignac, and Parvez Faruki. Network intrusion detection for iot security based on learning techniques. *IEEE Communications Surveys & Tutorials*, 21(3):2671–2701, 2019.

[18] Ibraheem Aljamal, Ali Tekeoğlu, Korkut Bekiroglu, and Saumendra Sengupta. Hybrid intrusion detection system using machine learning techniques in cloud computing environments. In *2019 IEEE 17th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 84–89, 2019.

[19] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[20] Congyuan Xu, Jizhong Shen, and Xin Du. A method of few-shot network intrusion detection based on meta-learning framework. *IEEE Transactions on Information Forensics and Security*, 15:3540–3552, 2020.

[21] Wei Liang, Yiyong Hu, Xiaokang Zhou, Yi Pan, I Kevin, and Kai Wang. Variational few-shot learning for microservice-oriented intrusion detection in distributed industrial iot. *IEEE Transactions on Industrial Informatics*, 18(8):5087–5095, 2021.

[22] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.

[23] Nsl-kdd dataset. http://nsl.cs.unb.ca/NSL-KDD/.

[24] Yingwei Yu and Naizheng Bian. An intrusion detection method using few-shot learning. *IEEE Access*, 8:49730–49740, 2020.

[25] Jingcheng Yang, Hongwei Li, Shuo Shao, Futai Zou, and Yue Wu. Fs-ids: A framework for intrusion detection based on few-shot learning. *Computers & Security*, 122:102899, 2022.

[26] Zu-Min Wang, Ji-Yu Tian, Jing Qin, Hui Fang, and Li-Ming Chen. A few-shot learning-based siamese capsule network for

intrusion detection with imbalanced training data. *Computational intelligence and neuroscience*, 2021, 2021.

[27] Tao Wu, Honghui Fan, Hongjin Zhu, Congzhe You, Hongyan Zhou, and Xianzhen Huang. Intrusion detection system combined enhanced random forest with smote algorithm. *EURASIP Journal on Advances in Signal Processing*, 2022(1):1–20, 2022.

[28] Shuokang Huang and Kai Lei. Igan-ids: An imbalanced generative adversarial network towards intrusion detection system in adhoc networks. *Ad Hoc Networks*, 105:102177, 2020.

[29] Aniwat Phaphuangwittayakul, Yi Guo, and Fangli Ying. Fast adaptive meta-learning for few-shot image generation. *IEEE Transactions on Multimedia*, 24:2205–2217, 2022.

[30] Aimin Yang, Chaomeng Lu, Jie Li, Xiangdong Huang, Tianhao Ji, Xichang Li, and Yichao Sheng. Application of meta-learning in cyberspace security: A survey. *Digital Communications and Networks*, 2022.

[31] Muhammad Usama, Muhammad Asim, Siddique Latif, Junaid Qadir, and Ala-Al-Fuqaha. Generative adversarial networks for launching and thwarting adversarial attacks on network intrusion detection systems. *2019 15th International Wireless Communications and Mobile Computing Conference, IWCMC 2019*, pages 78–83, 2019.

[32] Miao Xie, Bingli Liu, Lu Wang, Cheng Li, Yunhui Kong, and Rui Tang. Auto encoder generative adversarial networks-based mineral prospectivity mapping in lhasa area, tibet. *Journal of Geochemical Exploration*, 255:107326, 2023.

[33] Canadian Institute for Cybersecurity. Csecic-ids2018 on aws. https://www.unb.ca/cic/datasets/ids-2018.html.

[34] Nickolaos Koroniotis, Nour Moustafa, Elena Sitnikova, and Benjamin Turnbull. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems*, 100:779–796, 2019.

[35] Basheer Husham Ali, Nasri Sulaiman, SAR Al-Haddad, Rodziah Atan, and Siti Lailatul Mohd Hassan. Ddos detection using active and idle features of revised cicflowmeter and statistical approaches. In *2022 4th International Conference on Advanced Science and Engineering (ICOASE)*, pages 148–153. IEEE, 2022.

[36] Node-red tool. https://nodered.org/.

[37] Argus tool. https://qosient.com/argus/index.shtml.

[38] Agus Eko Minarno, Laofin Aripa, Yufis Azhar, and Yuda Munarko. Classification of malaria cell image using inception-v3 architecture. *JOIV: International Journal on Informatics Visualization*, 7(2):273–278, 2023.

[39] Md Hasan Shahriar, Nur Imtiazul Haque, Mohammad Ashiqur Rahman, and Miguel Alonso. G-ids: Generative adversarial networks assisted intrusion detection system. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 376–385. IEEE, 2020.

[40] Bin Tang, Yan Lu, Qi Li, Yueying Bai, Jie Yu, and Xu Yu. A diffusion model based on network intrusion detection method for industrial cyber-physical systems. *Sensors*, 23(3):1141, 2023.

[41] Koorosh Aslansefat, Ioannis Sorokos, Declan Whiting, Ramin Tavakoli Kolagari, and Yiannis Papadopoulos. Safeml: safety monitoring of machine learning classifiers through statistical difference measures. In *International Symposium on Model-Based Safety and Assessment*, pages 197–211. Springer, 2020.

[42] Mohamed Hammad, Nabil Hewahi, and Wael Elmedany. Mmm-rf: A novel high accuracy multinomial mixture model for network intrusion detection systems. *Computers & Security*, 120:102777, 2022.

[43] Alper Sarıkaya, Banu Günel Kılıç, and Mehmet Demirci. Gru-gbm: A combined intrusion detection model using lightgbm and gated recurrent unit. *Expert Systems*, 39(9):e13067, 2022.

[44] Erik Miguel de Elias, Vinicius Sanches Carriel, Guilherme Werneck De Oliveira, Aldri Luiz Dos Santos, Michele Nogueira, Roberto Hirata Junior, and Daniel Macêdo Batista. A hybrid cnn-lstm model for iiot edge privacy-aware intrusion detection. In *2022 IEEE Latin-American Conference on Communications (LATINCOM)*, pages 1–6, 2022.

[45] Emil Selvan GSR, M Azees, CH Rayala Vinodkumar, and G Parthasarathy. Hybrid optimization enabled deep learning technique for multi-level intrusion detection. *Advances in Engineering Software*, 173:103197, 2022.

[46] Riccardo Lazzarini, Huaglory Tianfield, and Vassilis Charissis. A stacking ensemble of deep learning models for iot intrusion detection. *Knowledge-Based Systems*, 279:110941, 2023.

[47] Ning Wang, Yimin Chen, Yang Hu, Wenjing Lou, and Y. Thomas Hou. Manda: On adversarial example detection for network intrusion detection system. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021.

[48] Simon Msika, Alejandro Quintero, and Foutse Khomh. SIGMA : Strengthening IDS with GAN and Metaheuristics Attacks. pages 1–11, 2019.

[49] Matthias Schonlau and Rosie Yuyan Zou. The random forest algorithm for statistical learning. *The Stata Journal*, 20(1):3–29, 2020.

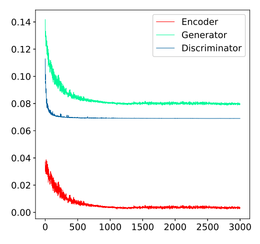
[50] Zilong Lin, Yong Shi, and Zhi Xue. Idsgan: Generative adversarial networks for attack generation against intrusion detection. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 79–91. Springer, 2022.

[51] Miel Verkerken, Laurens D'hooge, Didik Sudyana, Ying-Dar Lin, Tim Wauters, Bruno Volckaert, and Filip De Turck. A novel multi-stage approach for hierarchical intrusion detection. *IEEE Transactions on Network and Service Management*, 2023.

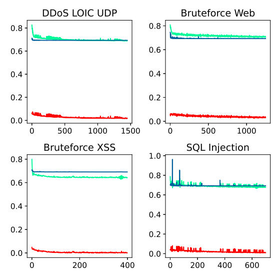
Appendix

TABLE A.1 Top five important features selected by RandomForestRegressor in each attack type

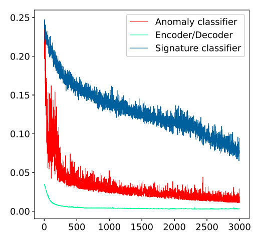
Attack type	Top five feature	Weight	Attack type	Top five feature	Weight	Attack type	Top five feature	Weight
Bot (CCI)	Flow Pkts/s	0.1029	DDoS LOIC HTTP (CCI)	Fwd Pkt Len Max	0.1153	FTP Bruteforce (CCI)	Fwd Seg Size Min	0.1286
	Fwd Pkts/s	0.0889		Fwd Seg Size Avg	0.1013		Bwd Pkts/s	0.1081
	Pkt Size Avg	0.0866		Fwd Pkt Len Mean	0.0991		Flow Pkts/s	0.0948
	Init Fwd Win Byts	0.0683		Flow Pkts/s	0.0898		Fwd Pkts/s	0.0851
	Bwd Seg Size Avg	0.0670		Fwd Pkts/s	0.0740		Init Fwd Win Byts	0.0765
SSH Bruteforce (CCI)	Fwd Seg Size Min	0.1809	DoS GoldenEye (CCI)	Fwd Seg Size Min	0.2064	DoS Slowloris (CCI)	Fwd Seg Size Min	0.1406
	Init Bwd Win Byts	0.1260		Fwd Header Len	0.0854		Bwd Pkt Len Mean	0.0562
	Bwd Pkts/s	0.0883		Init Fwd Win Byts	0.0633		Bwd Seg Size Avg	0.0559
	Bwd Header Len	0.0680		Flow Pkts/s	0.0555		Bwd Pkt Len Max	0.0545
	Fwd Pkts/s	0.0600		Fwd Pkts/s	0.0503		Flow IAT Std	0.0467
DoS Hulk (CCI)	Fwd Seg Size Min	0.1510	DoS SlowHTTPTest (CCI)	Fwd Seg Size Min	0.1291	DDoS HOIC (CCI)	Init Fwd Win Byts	0.2271
	Fwd Header Len	0.0619		Bwd Pkts/s	0.1069		Fwd Header Len	0.0880
	Subflow Bwd Pkts	0.0584		Flow Pkts/s	0.0923		Fwd Pkts/s	0.0772
	Tot Bwd Pkts	0.0570		Fwd Pkts/s	0.0837		Flow Pkts/s	0.0703
	Bwd Header Len	0.0496		Init Fwd Win Byts	0.0770		Tot Fwd Pkts	0.0662
Infiltration (CCI)	Pkt Len Std	0.0780	DDoS LOIC UDP (CCI)	Fwd Act Data Pkts	0.1191	Bruteforce Web (CCI)	Init Fwd Win Byts	0.1796
	Init Fwd Win Byts	0.0494		TotLen Fwd Pkts	0.1150		Fwd Seg Size Avg	0.0723
	Flow Pkts/s	0.0442		Subflow Fwd Byts	0.1121		Fwd Pkt Len Mean	0.0698
	Fwd Pkt Len Std	0.0413		Fwd Header Len	0.1056		TotLen Fwd Pkts	0.0451
	Fwd Pkts/s	0.0404		Subflow Fwd Pkts	0.1051		Subflow Fwd Byts	0.0445
Bruteforce XSS (CCI)	Init Fwd Win Byts	0.1832	SQL Injection (CCI)	Init Fwd Win Byts	0.1618	DoS TCP (BI)	ltime	0.1763
	Flow Pkts/s	0.0526		Flow Pkts/s	0.0868		N_IN_Conn_P_DstIP	0.1566
	Fwd Pkts/s	0.0519		Fwd Pkts/s	0.0838		N_IN_Conn_P_SrcIP	0.1274
	Fwd Header Len	0.0513		Pkt Len Max	0.0810		Pkts_P_State_P_Protocol_P_DestIP	0.0962
	Subflow Fwd Pkts	0.0472		Bwd Pkt Len Std	0.0784		TnBPDstIP	0.0748
DoS UDP (BI)	ltime	0.1721	DDoS TCP (BI)	ltime	0.1892	DDoS UDP (BI)	N_IN_Conn_P_DstIP	0.1647
	stddev	0.1461		N_IN_Conn_P_DstIP	0.1631		ltime	0.1615
	N_IN_Conn_P_SrcIP	0.1152		TnP_PDstIP	0.1159		sum	0.0957
	N_IN_Conn_P_DstIP	0.1148		TnBPDstIP	0.1006		Pkts_P_State_P_Protocol_P_DestIP	0.0939
	sum	0.0790		TnP_Per_Dport	0.0919		mean	0.0885
Service Scan (BI)	ltime	0.2481	OS Fingerprint (BI)	ltime	0.2566	DoS HTTP (BI)	ltime	0.1986
	TnP_PerProto	0.1679		N_IN_Conn_P_DstIP	0.1596		drate	0.1269
	TnP_Per_Dport	0.1133		TnP_PDstIP	0.1137		TnP_Per_Dport	0.1137
	dur	0.0822		TnP_PerProto	0.1082		Pkts_P_State_P_Protocol_P_DestIP	0.1021
	N_IN_Conn_P_DstIP	0.0600		TnBPDstIP	0.0907		TnP_PDstIP	0.0684
DDoS HTTP (BI)	N_IN_Conn_P_DstIP	0.1742	Keylogging (BI)	ltime	0.2280	Data Exfiltration (BI)	ltime	0.2358
	ltime	0.1709		AR_P_Proto_P_Dport	0.1307		drate	0.0746
	drate	0.1032		AR_P_Proto_P_Sport	0.1103		TnP_PerProto	0.0574
	TnP_PDstIP	0.0917		N_IN_Conn_P_DstIP	0.0987		N_IN_Conn_P_DstIP	0.0540
	TnBPDstIP	0.0717		AR_P_Proto_P_SrcIP	0.0806		sbytes	0.0442



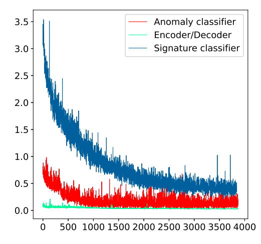
(a) Algorithm 1 on CCI



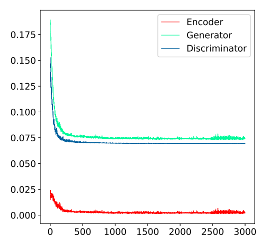
(b) Algorithm 2 on CCI



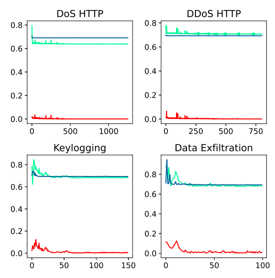
(c) Algorithm 3 on CCI



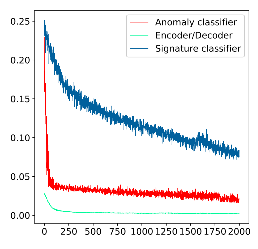
(d) Algorithm 4 on CCI



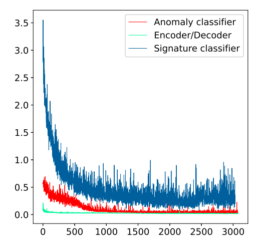
(e) Algorithm 1 on BI



(f) Algorithm 2 on BI



(g) Algorithm 3 on BI



(h) Algorithm 4 on BI

Fig. A.1 Training losses variation with batches: (a), (b), (c), (d) is based on CSE-CIC-IDS2018 while (e), (f), (g), (h) is based on Bot-IoT; the curves in (a), (e), (b), (f) refer to losses of E_p, G, Dis in Algorithm 1 and 2, respectively; the curves in (c), (g), (d), (h) refer to losses of $E_f/D_f, C_a, C_s$ in Algorithm 3 and 4, respectively

TABLE A.2 Acc comparison on signature-based classification for few-shot attacks in CCI under different M_s and A (%)

Training samples	Generated amount A				
	0	10	100	1000	10000
$M_s = 5$	73.2	77.4	80.7	90.1	92.5
$M_s = 10$	75.5	78.1	81.8	91.6	93.6
$M_s = 20$	78.3	80.2	83.5	93.4	95.6
M_s in Table 4	81.7	83.2	86.7	95.2	97.3

TABLE A.3 Acc comparison on signature-based classification for few-shot attacks in BI under different M_s and A (%)

Training samples	Generated amount A				
	0	10	100	1000	10000
$M_s = 5$	71.6	73.8	77.2	83.6	85.4
$M_s = 10$	77.4	77.2	77.9	88.7	82.7
$M_s = 20$	79.5	79.1	80.0	86.1	89.1
M_s in Table 4	76.9	82.8	83.8	91.9	92.4

TABLE A.4 Acc comparison on signature-based classification for few-shot and all attacks under different A (%)

Target	Generated amount A				
	0	10	100	1000	10000
Few-shot (CCI)	81.7	83.2	86.7	95.2	97.3
All (CCI)	82.6	83.3	85.1	**91.4**	89.4
Few-shot (BI)	76.9	82.8	83.8	91.9	92.4
All (BI)	90.9	93.1	93.9	**96.3**	94.3

TABLE A.5 Average training time and FID comparison on attack generation with generative-based methods in CCI

Method	Time (s)	FID(↓)	Epoch
GAN-enhanced DNN[9]	82	42.5	50
	172	31.3	100
G-IDS[39]	87	110.5	50
	124	64.3	100
DDPM[40]	62	267.3	50
	128	237.3	100
G in MAGET	**51**	**23.0**	50

TABLE A.6 Average training time and FID comparison on attack generation with generative-based methods in BI

Method	Time (s)	FID(↓)	Epoch
GAN-enhanced DNN[9]	23.6	28.8	50
	56.4	20.3	100
G-IDS[39]	19.7	53.5	50
	42.8	35.8	100
DDPM[40]	67	159.5	50
	137	140.5	100
G in MAGET	**16**	**4.1**	50

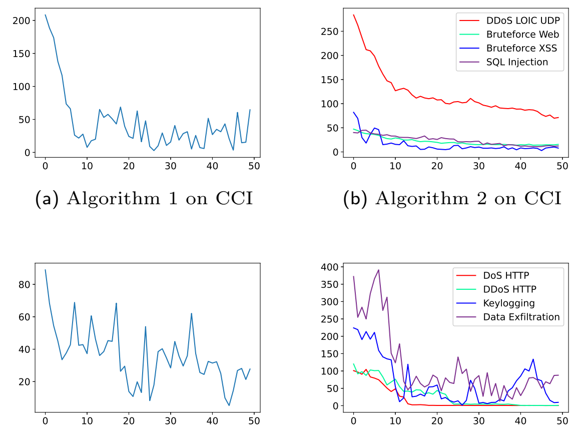
TABLE A.7 Comparison on sub-modules and combined one with an advanced method in CCI and BI (%)

Method	sub-module	Acc	F1-Score
Multi-Stage[51]	Anomaly*	96.5/99.3	96.5/99.3
MAGET	C_a	96.8/99.4	97.0/99.6
Multi-Stage[51]	Multi-class	88.4/90.1	86.5/89.2
MAGET	C_s	**91.4/96.3**	**92.2/96.1**
Multi-Stage[51]	combined	91.8/96.7	90.2/94.9
MAGET	combined	**95.1/98.9**	**94.8/99.1**

* The anomaly detection stage uses an extension stage to support its benign traffic filtering.

TABLE A.8 Ablation experiment of MAGET on CCI and BI

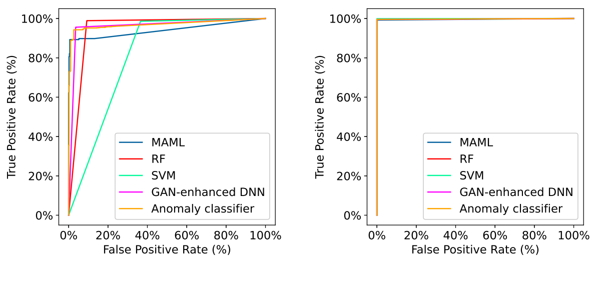
Method	FID	TPR (%)	FPR (%)	Acc (%)
No pre-training	31.3/20.3	95.6/99.7	3.6/0.1	91.7/89.0
No auto-encoders	45.4/33.7	91.8/99.6	2.1/0.3	90.5/88.8
No generation	-	89.3/99.8	0.6/0.1	85.3/84.6
MAGET	**23.0/4.1**	94.3/**99.8**	1.8/**0.1**	**95.2/91.9**



(a) Algorithm 1 on CCI (b) Algorithm 2 on CCI

(c) Algorithm 1 on BI (d) Algorithm 2 on BI

Fig. A.2 FID variations of the generative attack instances with epochs: (a), (b) is based on CSE-CIC-IDS2018 while (c), (d) is based on Bot-IoT; (a), (c) are for Algorithm 1 and (b), (d) are for Algorithm 2



(a) C_a on CCI (b) C_a on BI

Fig. A.3 ROC curves on anomaly-based classification with baseline methods