Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Federated learning for reliable model updates in network-based intrusion detection

Roger R. dos Santos [a], Eduardo K. Viegas [a,*], Altair O. Santin [a], Pietro Tedeschi [b]

[a] Pontifical Catholic University of Paraná, Curitiba, Paraná, 80215-901, Brazil
[b] Autonomous Robotics Research Center, Technology Innovation Institute, Abu Dhabi, United Arab Emirates

ARTICLE INFO

Dataset link: https://secplab.ppgia.pucpr.br/reliablefedlearningids

Keywords:
Intrusion detection
Network attacks
Federated learning
Reject option
Reliability

ABSTRACT

Machine Learning techniques for network-based intrusion detection are widely adopted in the scientific literature. Besides being highly variable, network traffic behavior changes over time, demanding proposed schemes to be periodically updated to ensure their reliability. Unfortunately, their efficiency is significantly limited in production environments. This paper proposes a new Federated Learning model for reliable network-based intrusion detection with highly confident model updates over time. Our proposed scheme assesses the classification reliability in an unsupervised fashion and rejects potential misclassifications even when outdated. In addition, it significantly eases the model update cost by conducting it in a Federated Learning rationale. To evaluate the effectiveness of our solution, we conduct an experimental campaign with a new dataset, MAWIFlow, with over 7 TB of real network traffic spanning a year. The achieved results of our proposed model are striking. It respectively improves the average false-positive and false-negative rates by up to 12% and 9.6% when no model updates are conducted. If done so, it can further improve the false-positive rate by up to 13% while rejecting only 3.6% of events and demanding only 0.3% of events for model updates. Further, the comparison against the traditional Federated Learning approach confirms our model's remarkable performance in several scenarios. Finally, the quality and viability of our solution do prove that our approach can be successfully adopted for improving the accuracy and efficiency of classification systems in real-world scenarios where outdated models are prevalent and pave the way for future research in the area.

1. Introduction

Despite the extensive research effort on developing new security solutions, the number of reported cyberattacks continues to increase as time passes. For example, according to a security report, in 2022 (Statistics, 2023), over 15% of Internet users have been targeted by a cyberattack. In general, to detect this increasing number of threats, operators resort to Network-based Intrusion Detection Systems (NIDSs), implemented through *misuse-based* or *behavior-based* approaches (Molina-Coronado et al., 2020). On the one hand, *misuse-based* techniques identify misbehavior according to a database of previously known threats, hence, being unable to address new kinds of attacks (Viegas et al., 2019). On the other hand, *behavior-based* techniques recognize misconducts according to the similarity to a previously modeled attack behavior. As a result, the latter can detect new attacks as long as they behave similarly to the previously modeled behavior (Zoppi et al., 2023).

In recent years, several works proposed new *behavior-based* NIDS through Machine Learning (ML) techniques to address the increasing number of cyberattacks (Molina-Coronado et al., 2020). To this aim, a behavioral ML model is built by evaluating the data available in a training dataset. To achieve optimal performance in the production environment, the training dataset must be composed of massive amounts (e.g., up to millions) of both normal and malicious network samples (Ramkumar et al., 2022).

In practice, the accuracy reliability of designed ML-based NIDSs relies on the training data quality. It must adequately reflect the network properties of the production environment, including the behavior of the deployed network services, the characteristics of the network link, and even the expected attacks that will be experienced when the system is used in production (Al-Hadhrami and Hussain, 2020). Indeed, building a reliable training dataset for NIDSs is a complex and time-consuming task (Kilincer et al., 2021). Notwithstanding, to guarantee

* Corresponding author.
E-mail addresses: roger.santos@ppgia.pucpr.br (R.R. dos Santos), eduardo.viegas@ppgia.pucpr.br (E.K. Viegas), santin@ppgia.pucpr.br (A.O. Santin), pietro.tedeschi@tii.ae (P. Tedeschi).

https://doi.org/10.1016/j.cose.2023.103413
Received 13 April 2023; Received in revised form 23 June 2023; Accepted 26 July 2023
Available online 1 August 2023
0167-4048/© 2023 Published by Elsevier Ltd.

that the designed ML-based NIDSs can detect a wide range of attack behaviors, the training dataset must present a considerable mixture of available network attacks. As a result, to adequately address such a challenge, ideally, network operators must have access to the training data adopted by multiple organizations (Campos et al., 2022). However, apart from the dataset-building challenges, collecting multiple network-traffic datasets is difficult due to the privacy concerns that arise when the participant organizations' Cyber Threat Intelligence (CTI) is publicly shared (Mills et al., 2022).

The behavior of network traffic is highly variable and non-stationary as time passes, a situation caused due to the provision of new kinds of services, the discovery of new categories of attacks, and even changes in the network link(s) (Zoppi et al., 2023). As a result, even if the network operator can build a "*perfect*" intrusion dataset, it cannot depict the non-stationary nature of network traffic behavior as time passes (Li et al., 2022). In such a case, to preserve the reliability of the designed systems, it is essential to conduct periodic model updates to the deployed ML-based NIDSs (Viegas et al., 2019) Indeed, an outdated ML model will produce a higher rate of false alarms over time, considering that the current behavior of the monitored environment does not reflect those used during the model-building procedure.

The model update is a challenging task that requires the availability of an updated and labeled training dataset and the execution of a resource-intensive model training process (Thakkar and Lohiya, 2021). On the one hand, as organizations cannot share CTI due to privacy concerns, operators typically require unfeasible amounts of time to collect and label the newly occurring network traffic on their own (Viegas et al., 2019). On the other hand, the traditional ML update procedure discards the outdated model to build a new one based on the newly built dataset without considering the prior knowledge acquired from the previous training (Li et al., 2021). As a result, the model update task in NIDSs often demands several weeks or even months to be conducted, leaving systems unprotected from new kinds of threats.

An outdated ML model cannot provide the same level of reliability as those measured during the testing phase. The system's false alarm rate increases, demanding the network operator to suppress further alerts. As model updates are time-demanding, designed systems must keep their reliability even when outdated while the building of a new ML model is still ongoing. Yet, most of the current approaches in the literature neglect the system reliability as time passes, assuming that a new ML model can be promptly provided (Lee et al., 2022). As a result, as current techniques often become unreliable shortly after their deployment, they are rarely deployed in production environments, remaining primarily as a research topic (Arp et al., 2021).

Contribution. In light of this, this paper proposes a new Federated Learning (FL) model that provides a reliable network-based intrusion detection over time, implemented in three phases. First, we perform the classification task with a reject option, which ensures that only highly confident, thus, more likely to be correctly classified samples are accepted by our system, even if the deployed model is outdated. Our main insight is to assess the quality of the performed classifications to ensure that our system only triggers alerts on highly confident network events, rejecting low-confident classifications. Second, we conduct the model training task through a FL scheme based on the rejected events over time. As a result, organizations can share their CTI while keeping their used training dataset private, easing the model training process and improving the built model's quality. Third, model updates over time are implemented in a FL fashion based on the rejected events while leveraging the outdated deployed model following a transfer learning rationale. Therefore, the model update task is significantly eased, since we consider fewer events for the model building and use the prior model knowledge. The insight of such an approach is that FL can relax the model update task by sharing the organizations' CTI while proactively selecting the new network traffic through the rejection approach.

Table 1
Features set extracted at the network level for each feature grouping in a time window interval of 15 s.

Grouping Features	Collected Features
SRC IP Addresses, SRC/DST IP Addresses, SRC/DST Service Ports	Number of Packets
	Number of Bytes
	Percentage of Packets (SYN Flag)
	Percentage of Packets (ACK Flag)
	Percentage of Packets (RST Flag)
	Percentage of Packets (FIN Flag)
	Percentage of Packets (CWR Flag)
	Percentage of Packets (URG Flag)
	Average Packet Size
	Percentage of Packets (ICMP Redirect Flag)
	Percentage of Packets (ICMP Time Exceeded Flag)
	Percentage of Packets (ICMP Unreachable Flag)
	Percentage of Packets (ICMP Other Types Flag)

In summary, the main contributions of this paper are:

- An evaluation of the classification performance over time of state-of-the-art ML-based NIDSs. Our experiments, performed through a dataset of over 7 TB of real network traffic spanning a year, show that currently used approaches in the literature demand unfeasible periodic model updates to be conducted.
- A new FL model aiming at a reliable network-based intrusion detection. Our proposed model can improve the false-positive rate by up to 12.9% when it is not periodically updated. In addition, model updates can be made using only 3.6% of instances while reaching a false-positive of only 3% in a FL manner without requiring organizations to share their sensitive training data.

Roadmap. The remainder of this paper is organized as follows. Section 2 further describes the ML-based NIDS challenges. Section 3 presents related work on reliable intrusion detection. Section 4 evaluates how network traffic behavior changes affect traditional ML-based NIDSs. Section 5 describes our proposed model. Section 6 evaluates our proposed scheme, and Section 7 concludes our work.

2. Preliminaries

This section further discusses the challenging aspects towards the design of a reliable ML-based NIDS.

2.1. Machine learning for network-based intrusion detection

In general, ML for NIDS is achieved through the implementation of four sequential modules (Molina-Coronado et al., 2020), namely:

1) *Data Acquisition*. Collect passing network events from the monitored environment, e.g., collecting network packets from a given Network Interface Card (NIC).
2) *Feature Extraction*. Evaluate the collected data to compound a feature vector that adequately describes the behavior of the collected event. In general, network events in NIDS are represented through network flows, which summarize the communication history between the network entities in a given time window. Table 1 shows an example of a feature set that can be used to build a network flow.
3) *Classification*. Classifies the extracted feature set as either *normal* or *attack* through the application of a ML model.
4) *Alert*. Signals to the network operator network events classified as *attack* by the ML model.

Over the last years, several highly accurate and promising ML-based approaches were proposed for the classification task, wherein authors typically resort to pattern recognition techniques (Zhang et al., 2022). To this aim, proposed schemes are implemented following a three-phase process, namely *training*, *validation*, and *testing*. The *training* phase goal consists of developing a ML model by evaluating a specified training dataset. To adequately perform such a task, the training dataset must contain huge amounts of network samples that are expected to be similar to those evidenced in production environments. The validation phase assists with the ML model fine-tuning task, such as performing feature selection and tuning the model hyper-parameters. Finally, at the testing phase, the model accuracy is estimated during the system deployment in the production environment.

2.2. On the challenge of realistic network traffic

The efficiency of a ML model relies on the use of realistic training data. Unfortunately, providing a training dataset that adequately represents the network environment behavior in production environments is not a trivial task (Zhang et al., 2022). On the contrary, the behavior of networked environments is highly variable and also changes as time passes. In practice, a realistic training dataset used for ML-based NIDSs building should provide the following characteristics:

- *Real network traffic.* The network traffic must include realistic communication between the clients and the network service(s). The exchanged network traffic between the network entities must follow the behavior in production environments.
- *Valid network protocols.* The available network traffic must strictly adhere to the network protocol specification as expected in production environments.
- *Diverse behavior.* The network traffic behavior must be highly mixed according to the selected network protocols on the dataset. Network traffic behavior in production environments varies significantly over time.
- *Realistic attack.* The attacker's behavior must be adequately represented in the dataset. The generated attacks must vary in their behavior and the attack type.
- *Previously labeled.* The collected network traffic must be adequately labeled as either *normal* or *attack*. The training task requires previously known labels associated with the input events.
- *Publicly available.* The built dataset must be publicly available to enable researchers to benchmark their proposed schemes.
- *Non-stationary behavior.* The behavior of network traffic changes as time passes. Therefore, the collected data must depict those changes.

As a result, building realistic intrusion datasets for ML-based NIDSs is challenging. Network operators generally build the training dataset in a controlled environment or collect it from their infrastructure (Yang et al., 2022). On the one hand, a controlled testbed usually fails to provide realistic and diverse network traffic behavior (Maseer et al., 2021). On the other hand, monitoring a network infrastructure provides realistic network traffic. However, it must address the labeling of collected events and can not be easily shared publicly (Papadogiannaki and Ioannidis, 2021).

The adequate labeling of the collected network traffic poses a great challenge to the reliability of designed intrusion datasets (Yang et al., 2022). A controlled testbed paves the way for automatically labeling network traffic, as the host's behavior can be previously known (Yamin et al., 2020). In contrast, labeling real-world network traffic collected from production environments is not easily achievable, wherein authors usually resort to misuse-based or unsupervised learning techniques (Fontugne et al., 2010). On the one hand, the prior enables labeling previously known attack behaviors while failing at identifying new attack variants. On the other hand, the latter is assumed to detect new attacks, usually at the expense of a high rate of false positives (Sommer and Paxson, 2010).

Apart from the challenges related to dataset building, the behavior of network traffic changes as time passes, a situation usually overlooked in the literature. In practice, the network traffic behavior evolves, which can be pushed by discovering new attacks, providing new services, or even changes on the network communication link (Viegas et al., 2019). As a result, deployed ML-based NIDS demands periodic model updates to be conducted. Generally, this resource and time-consuming task requires expert assistance for the data collection and labeling tasks (Zoppi et al., 2023).

2.3. Federated learning

The FL aims at conducting a collaborative learning process to enable information sharing between multiple and trusted peers (Hei et al., 2020). In practice, the FL goal is to perform the model training leveraging the participant peers' training data while keeping each peer-used data private. The FL scheme relies on a central server responsible for aggregating the local models into a global counterpart to achieve such a goal (Alghamdi and Bellaiche, 2023).

Given a classification function $f(x) : x \to y$ that outputs the identified label y on given an input feature vector x. The FL goal is to find an aggregated model w_G to be used by the function f, such that w_G is built based on the training data $\{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k\}$, where $\mathcal{D}_i$ denotes the private training dataset from peer P_i, and k represents the number of peers.

To achieve such a goal, FL is usually implemented following a four-phase process, namely *Initialization*, *Distribution*, *Local Training*, and *Aggregation*.

1) *Initialization.* The central server s initializes a selected global classification model w_G^t where t denotes the execution round. The selected model must enable the aggregation task to be conducted, e.g., using a neural network to aggregate its computed weights.
2) *Distribution.* At every communication round t, such that $0 \le t \le T$, where T denotes the upper limit of communication rounds, the central server s distributes the global model w_G^t to a selected number m of peers such that $0 \le m \le k$, where k denotes the total number of participant peers.
3) *Local Training.* Each previously selected peer P_i conducts the local training of the received global model w_G^t based on their private training data $\mathcal{D}_i$, compounding a local model w_i^t.
4) *Aggregation.* The central server s collects the local models from each selected peer to conduct the model aggregation task that compounds a new global model w_G^{t+1}. The model aggregation task is a function that aggregates a series of models into a single counterpart, e.g., through the Average Federated Learning (FedAVG) function (Zhou et al., 2021), implemented based on the following equation:

$$w_G^{t+1} = \frac{\sum_{i=0}^{m} w_i^t}{m} \tag{1}$$

where m denotes the number of selected peers. Therefore, the FedAVG build the new global model w_G^{t+1} by computing the average of each local model w_i^t weights.
Finally, if the execution round $t+1$ reaches the upper limit of rounds T, the training is terminated; otherwise, a new *Distribution* phase occurs.

As a result, the FL training procedure can build a robust ML model based on each organization's training data while keeping it private during such a process. The reason for this is that the training process is carried out locally based on each organization's private data, while the aggregation only requires sharing the built local models.

3. Related work

3.1. Intrusion detection reliability

Over the last decades, several works have proposed highly accurate ML techniques for network-based intrusion detection (Molina-Coronado et al., 2020; Yang et al., 2022). Despite the promising results reported in the scientific literature, proposed schemes are hardly deployed in production environments, remaining mainly as a research topic (Arp et al., 2021). In general, related works do not address the challenges of their proposed schemes when they need to be deployed in production, such as the evolving behavior of network traffic and the periodic model update procedure (Viegas et al., 2019). For example, Saba et al. (2022) presented a deep learning scheme for intrusion detection in resource-constrained devices. The proposed approach improves accuracy on an outdated dataset compared to other intrusion detection schemes. However, during their evaluation, they do not consider the network's highly variable and non-stationary behaviors. In contrast, Zeng et al. (2022) addressed the lack of reliability in intrusion detection by aiming for higher model generalization capabilities. The authors proposed a deep causal stable learning system to identify causal relationships between different intrusion datasets. Their model achieved more stable accuracy rates throughout different datasets, ensuring model generalization. Unfortunately, the authors consider the network traffic static without evaluating the changes in the causal relationship over time. Zhao et al. (2022) addressed non-stationary behavior in intrusion detection systems through an out-of-distribution detection approach. The authors build a system considering unknown behavior to their deployed model, enabling the identification of new events. Similarly, the authors do not consider (i) the changes in network traffic behavior as time passes and (ii) the model update challenge. Wahab (2022) highlighted changes in network traffic by resorting to a concept drift detection mechanism. The proposed model assumes a supervised setting, wherein the event label can be provided as time passes: a condition unrealistic in production environments.

3.2. Model updates

Several works in Academia like (Li et al., 2022) assume that the accuracy rate during the testing phase does not change over time. When considering model updates, researchers make use of incremental learning approaches. Mahdavi et al. (2022) proposed an active learning approach implemented through incremental model updates to address the evolving network traffic behavior. The authors identify new network traffic behavior according to previously known event clusters, assuming that new traffic will differ significantly. However, the network traffic behavior may not present significant variations over time, even for new behaviors. Similarly, Huang and Ma (2023) presented incremental learning in intrusion detection to merge new attack behaviors. Although their proposed model addresses the new type of attacks over time, it does not consider how the new network traffic behavior can be identified. Jiang et al. (2022) introduced incremental ensemble learning to improve prediction accuracy. The authors assume the easiness of signaling new traffic behaviors, without considering the challenge of new network traffic behavior identification. Wu et al. (2022) incrementally adjusted an ensemble of classifiers based on newly occurring data. Their approach can improve the classification accuracy of new data batches. Similarly, they do not take into account the identification of the new network traffic behavior. Li et al. (2021) addressed intrusion model updates through a transfer learning rationale. The authors use the outdated model to decrease the computational costs during model updates. Their proposed scheme do not consider the identification of new network traffic behavior.

3.3. Federated learning in intrusion detection

In the scientific literature, several FL approaches have been proposed to address challenges in the network-based intrusion detection context (Campos et al., 2022). For instance, Yuan et al. (2021) aimed at decreasing communication overheads while sharing the federated models. The authors proposed a model based on gradient boosting and decision trees to decrease communication costs while providing high accuracy. Unfortunately, the authors do not consider the model update procedure over time and how their model performs while facing new network traffic behavior. Another approach aiming the easiness during the model aggregation is proposed by Sun et al. (2020). Before building a globalized counterpart, the authors developed local models in each network segment. Although the proposed scheme can decrease the model aggregation costs, it does not manage the model update procedure and the classification reliability.

In general, FL has been used in intrusion detection to enable participants to share intrusion knowledge without compromising the participants' data privacy. Mothukuri et al. (2022) proposed an intrusion detection model through FL to enable resource-constrained devices to share local data knowledge without sharing data content. The proposed model achieves similar accuracies compared to globalized traditional approaches. However, the authors do not consider the detection reliability and the model update process over time. In contrast, Hei et al. (2020) aimed the application of blockchain to store the model local weights over time reliably. The authors proposed a scheme that can perform alert filtering without considering the model update procedure and its reliability.

3.4. Discussion

Network-based intrusion detection through *behavior-based* techniques is a widely explored topic in the literature (Molina-Coronado et al., 2020; Yang et al., 2022), wherein the vast majority of approaches focus their efforts on increasing the system accuracy (Saba et al., 2022; Zeng et al., 2022). Unfortunately, network traffic behavior is highly variable and evolves as time passes, demanding systems to be designed accordingly. Surprisingly, the classification reliability is hardly considered. It is assumed that new network traffic behavior can be easily identified, while the event label can be readily provided in production (Wahab, 2022; Zhao et al., 2022). On the contrary, event label is hardly available in reality, and designed techniques must operate in an unsupervised fashion.

To address the challenges related to detecting new network traffic behavior, researchers propose to adopt the incremental learning (Huang and Ma, 2023; Jiang et al., 2022; Mahdavi et al., 2022) or transfer learning (Li et al., 2021) approaches which ease the model update computational costs. However, (i) the detection reliability before the availability of the newly built model and (ii) the identification of the new network traffic is not correctly considered (Li et al., 2021; Wu et al., 2022). In such a context, FL-based approaches can ease the data requirements during model updates, such as solving the communication overheads (Sun et al., 2020; Yuan et al., 2021) or decreasing the computational costs (Mothukuri et al., 2022).

As a result, most of the current ML-based intrusion detection schemes cannot address the challenges of real-world networked environments due to the high variability of network traffic and the model update procedure to address new network traffic behaviors.

4. Problem statement

Over the last years, proposed ML-based intrusion detection schemes did not consider the challenges related to the evolving behavior of network traffic. In this section, we investigate how the changes in network traffic behavior over time may affect the classification performance of designed ML-based approaches. We introduce the dataset adopted in

Table 2
MAWIFlow dataset statistics.

Property	Value
Average Daily Network Packets	105 Millions
Average Daily Network Flows	19 Millions
Average Daily Throughput	610 Mbps
Average Daily Anomalous Flows	1.8 Millions
Average Daily Dataset Size	19.7 GB
Total Network Packets	27.72 Billions
Total Network Flows	6.14 Billions
Total Dataset Size	7.1 TB

our work to achieve such a goal, and then we experimentally evaluate the classification performance when the network traffic behavior changes over time.

4.1. MAWIFlow dataset

The design of a realistic network-based intrusion detection dataset is challenging because the features needed during the dataset building are not readily achievable (see Sec. 2.2). Current approaches in the literature assume only a static network traffic behavior without accounting for its changes, i.e., they leveraged the same training and testing dataset environment (Gates and Taylor, 2006). In practice, using outdated datasets with several known flaws is common.

Our work resorts to the *MAWIFlow* dataset to address such a challenge. The dataset comprises one year of real network traffic collected daily in a 15 minute interval. The network traffic is collected from a transit link connecting Japan and USA, as provided by the MAWI network traffic archive (MAWI, 2023). For our evaluation, we select the network traffic that occurred through 2014, due to a higher available network data samples. The resulting dataset comprises ≈ 7 TB of data with over 6 billion network flows. The collected network traffic shows a realistic behavior of production environments, with realistic network attacks experienced as time passes. The dataset presents a wide range of network-level attacks, such as *service scan*, *UDP scan*, *TCP scan*, *denial-of-service*, and even *web server* attacks.

We resort to an unsupervised ML technique to address the labeling of the collected network events. In detail, we label the *MAWIFlow* dataset events through MAWILab (Fontugne et al., 2010) technique, which labels the input events as either *normal* or *attack* in an unsupervised fashion. MAWILab uses several unsupervised ML algorithms to identify anomalous events that occurred on the MAWI archive daily without requiring human assistance. As discussed previously (see Sec. 2.2), unsupervised ML techniques may introduce false positives. To address such a challenge while accounting for the changes in network traffic behavior as time passes MAWILab builds its unsupervised ML algorithms daily. Therefore, as the dataset goal is to enable the evaluation of the reliability of designed ML-based NIDSs over time, the changes in the network traffic behavior will also be depicted on the produced dataset.

We label the identified anomalies as *attack* on *MAWIFlow* dataset, while the remaining events are considered as *normal* network traffic. To perform the feature extraction, we use the BigFlow (Viegas et al., 2019) tool, which extracts network flows in intervals of 15 seconds while extracting 40 flow-based features (Dromard et al., 2017) for each exported flow. Table 1 shows a summary of the extracted set of features in our dataset, while Table 2 shows the *MAWIFlow* dataset statistics.

The built dataset can adequately address the challenges related to a realistic intrusion dataset. The used network traffic is *real*, *valid*, and *highly diverse* as it is collected from a real-world network transit link. The available network attacks are *realistic* as the attack behavior is extracted from production environments. We address the *previous labeling* of events through an unsupervised technique, which can label the network events without human assistance. The network traffic is *non-stationary* as we consider a considerable recording period spanning an entire year. Finally, the dataset is provided in a *publicly available* format. It does not contain sensitive information because MAWI removes the network packet payload and anonymizes the network packet headers. As a result, our dataset provides a benchmark to evaluate proposed intrusion detection techniques concerning their reliability under real-world conditions of networked environments.

4.2. The reliability of ML-based NIDS

Due to the challenge of the impact related to the changes in the network traffic behavior over time on designed ML-based NIDSs, we evaluated the performance of widely used ML classifiers in the literature when applied on MAWIFlow dataset. Our experiments adopt four ML classifiers, namely Random Forest (RF), k-Nearest Neighbor (kNN), Bagging (Bag), and Ensemble (Zhang et al., 2022). The RF classifier uses 100 decision trees as its base learner, each using $gini$ as the node split quality metric. The kNN classifier is implemented with 5 neighbors as the k parameter while computing the distance between the events through the euclidean distance formula. The Bag classifier uses 10 decision trees as its base learner, with each learner trained on the entire training dataset with the samples randomly selected with replacement. The Ensemble classifier makes use of the three previously described classifiers (RF, kNN, and Bag) combined through a majority voting procedure. To ensure the proper model training, as most events on MAWIFlow dataset are normal (see Table 2), we apply a random undersampling without replacement at every training task. The classifiers are implemented through *scikit-learn* API v0.24 (scikit-learn, 2023).

We evaluate the selected classifiers using the following classification performance metrics:

- *True Positive (TP)*: number of attack samples correctly classified as an attack.
- *True Negative (TN)*: number of normal samples correctly classified as normal.
- *False Positive (FP)*: number of normal samples incorrectly classified as an attack.
- *False Negative (FN)*: number of attack samples incorrectly classified as normal.

Further, we measure the F-Measure according to the harmonic mean of precision and recall values while considering attack samples as positive and normal samples as negative, as shown in Eq. (4).

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

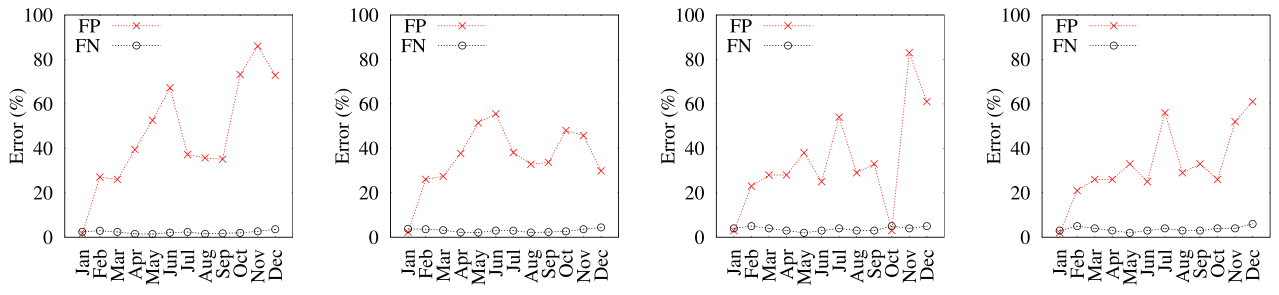
$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F\text{-}Measure = 2 \times \frac{Precision \cdot Recall}{Precision + Recall} \tag{4}$$

Our performed experiments aim at answering two Research Question (RQs):

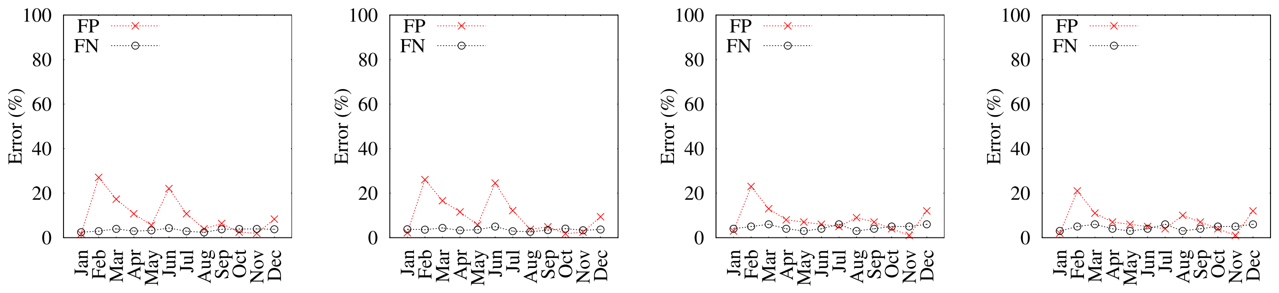
- (**RQ1**) *Is classification performance affected by network traffic behavior changes over time?*
- (**RQ2**) *Does model update address the network traffic behavior changes?*

Our first experiment aims at answering **RQ1**. In such a case, we evaluate the classification accuracy of the four selected classifiers over time without performing periodic model updates, as usually made in the literature. The four selected classifiers are trained based on the data collected in the first month of the MAWIFlow dataset (January) while measuring its accuracy throughout the year. As a result, we evaluate the classification performance as time passes if no model updates are performed periodically. Although the network traffic behavior is expected to change over time, the impact on classification accuracy of designed ML-based techniques is often not considered.



(a) Random Forest (RF) (b) k-Nearest Neighbor (kNN) (c) Bagging (Bag) (d) Ensemble

Fig. 1. Monthly accuracy behavior of selected classifiers in *MAWIFlow* dataset without periodic model updates. Classifiers are trained using January data and not updated as time passes.



(a) Random Forest (RF) (b) k-Nearest Neighbor (kNN) (c) Bagging (Bag) (d) Ensemble

Fig. 2. Monthly accuracy behavior of selected classifiers in *MAWIFlow* dataset with monthly model updates. Classifiers are updated monthly with the data that occurred in the preceding month.

Fig. 1 shows the classification performance of the selected classifiers over time without periodic model updates. All the evaluated approaches can provide significantly high detection accuracies in January, i.e., the period in which they were trained. However, their measured error rates significantly increase after the training time. For example, just a month after the training period (February), the measured FP rate increases by 27%, 24%, 20%, and 21% for the RF, kNN, Bag, and Ensemble classifiers, respectively. Notwithstanding, if no model updates are performed, classification error can increase up to 87% (Fig. 1a, November). Even the best-performing classifier (Fig. 1d, Ensemble) significantly degrades its accuracy as time passes, presenting an average of 33% and 3% of FP and FN rates respectively, an increase of 31% and 1% when compared to its training period.

Accordingly, ML-based NIDSs must conduct periodic model updates to ensure their reliability. Our experiments show that the selected techniques must keep their classification reliability in the face of new network traffic behavior. However, there still needs to be more research on how the changes in network traffic over time affect designed techniques, despite, as evaluated, posing a significant challenge to designed approaches.

Our second experiment goal is to answer **RQ2** and experimentally verify if periodic model updates can provide detection reliability to ML-based NIDSs over time. To achieve such a task, we conduct monthly model updates according to the network data that occurred in the preceding month. For example, on March 1st, we update the model using the network flows that occurred throughout February as training data. We aim to reproduce a model update setting where the network operator conducts monthly model updates, considering a one-month model lifespan, regardless of the current network traffic behavior. In practice, the network operator cannot quickly assess when model updates should be conducted in production settings, as the label of events in network environments is not previously known.

Fig. 2 shows the classification performance of the selected classifiers when we conduct monthly model updates. The classifier's accuracies are measured according to the classification performance on a given month using the model trained on the data that occurred in the preceding month. In contrast to their no-update counterpart, the monthly updated classifier improved the classification accuracy significantly by keeping the error rates similar to those obtained in January. For example, in November, the RF classifier shows an FP rate of only 1.3%, as opposed to 87% compared to its no-update counterpart. In addition, the Ensemble classifier (Fig. 2d) achieved and average of 8% and 4% of FP and FN rates respectively. In practice, periodic model updates improved the classification accuracy of all selected classifiers, showing that model updates are necessary to address the network traffic behavior changes.

As a result, to keep their reliability, ML-based NIDSs must be periodically updated. Our experiments have shown a direct relationship between model reliability and model lifespan. Unfortunately, model updates are not easy to be conducted in NIDSs, thus, posing a great challenge for the deployment in production environments (see Sec. 2.1).

4.3. Discussion

The evolving behavior of network traffic poses a significant challenge to designed ML-based NIDSs, which can significantly affect the accuracy of used intrusion detection schemes. This section has experimentally evaluated how the changes in the network traffic behavior as time passes may affect the reliability of ML-based intrusion detection. The performed experiments have shown that the selected classifiers decreased their accuracy just a month after the training period (Fig. 1). In practice, current ML-based techniques significantly decrease their error rates as time passes. In such a context, model updates can address the changes in the network traffic if performed according to a given model lifespan (Fig. 2). However, there is a significant number of challenges to achieve this goal. Model updates require the provision of a new training dataset with properties that ensure its authenticity (see Sec. 2.2). Therefore, it must be executed without demanding significant efforts for the network operator while guaranteeing that the deployed model is reliable.

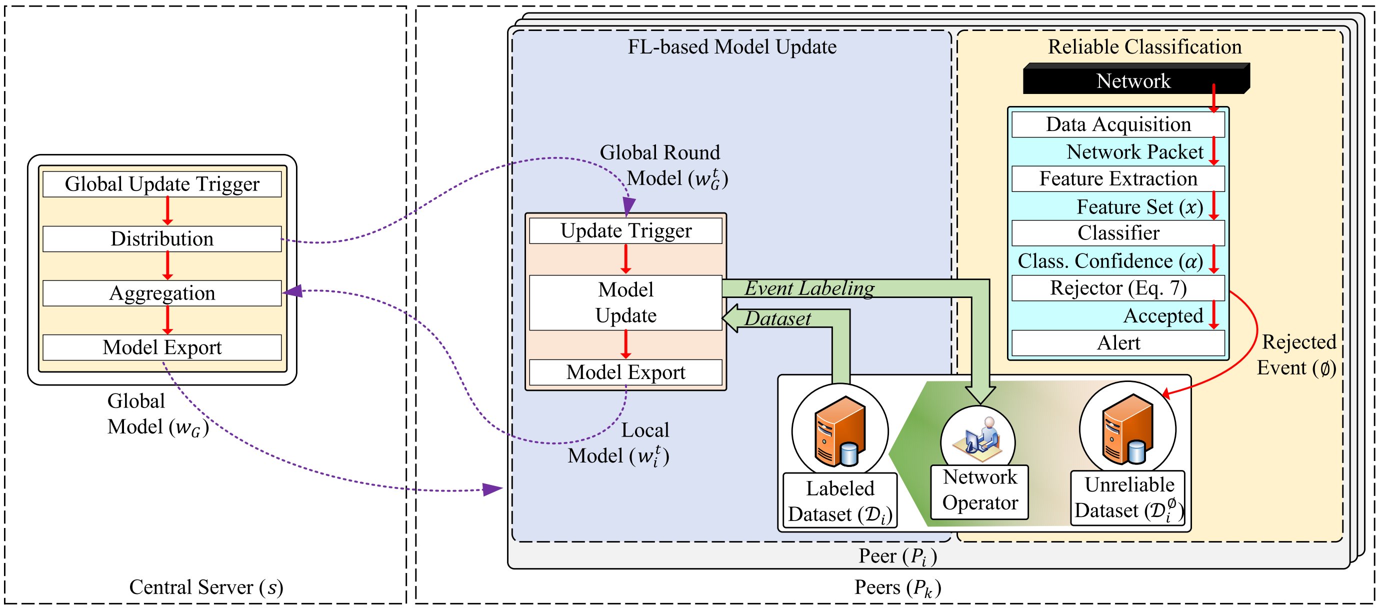


Fig. 3. Proposed federated learning model for reliable model updates on network-based intrusion detection.

5. A federated learning model for reliable model updates on network-based intrusion detection

This section describes our proposed reliable network-based intrusion detection model based on a FL strategy. Our solution addresses the evolving behavior of network traffic as time passes by conducting the classification with a reject option while leveraging model updates implemented following a FL rationale. The main proposal goal is to ease the network operator's burden to update the deployed intrusion detection model while keeping the system reliable. The proposed model is implemented in two phases, namely *Reliable Classification*, and *FL-based Model Update*, as depicted in Fig. 3.

We consider a FL architecture composed of a central server s and a set of peers P_k, where k denotes the number of peers such that $k > 1$. Each peer P_i is an organization that aims at conducting reliable intrusion detection while easing the model update procedure. We assume that the central server s is benign and trusted by all peers. The central server is responsible for compounding the global model w_G, executing the tasks of *initialization*, *distribution*, and *aggregation* (see Sec. 2.3). Conversely, each FL peer P_i executes the *local training* task based on their private labeled dataset $\mathcal{D}_i$.

Each peer executes the *Reliable Classification* module (Fig. 3) for conducting network-based intrusion detection on his infrastructure. Our proposal aims to achieve classification reliability by performing the classification with a reject option rationale. Our main goal is to identify unreliable classifications based on the confidence values output by the deployed classifiers as time passes. It is worth noticing that classification confidence is classifier agnostic. For instance, the RF classifier usually outputs its classification confidence values according to the ratio of individual decision trees that classify the input event for a given label. We assume that low-confident classifications have a higher probability of producing a false alarm; therefore, they should be rejected by our system. Rejected events are used twofold for alert suppression and model updates. Rejected events, due to their low classification confidence, do not generate alerts to the network operator, as they are more likely to produce a false alert. Nonetheless, rejected events are assumed to be new behaviors unknown to the deployed ML model, hence, they are stored for the following model updates. As a result, our scheme can keep its classification reliability as time passes while proactively selecting which events should be used for model updates.

To address the non-stationary behavior of network traffic, our proposal uses a *FL-based Model Update* module, which aims at easing the model update twofold. First, as time passes, each peer P_i conduct periodic model updates through a FL rationale by updating the global model w_G^t at each execution round t to build a global model w_G jointly. As a result, each participant organization can decrease the costs related to the model update task by sharing their CTI knowledge without affecting their privacy. The training data needed to conduct model updates can be significantly decreased as organizations share the knowledge obtained from their local data. Second, we conduct periodic model updates through a transfer learning approach by employing the FL rationale for each peer's rejected events $\mathcal{D}_i^\emptyset$. Thus, our proposed model can significantly decrease the computational costs during model updates as each peer collaborative rebuilds the outdated global model. These assumptions will reduce the data needed to conduct such a task by making the model update burden easy for the network operator.

The following subsections further describe our proposed FL model and the implementation details.

5.1. Reliable classification

The behavior of real-world network traffic is highly variable as time passes. This phenomenon affects the reliability of designed ML-based NIDSs schemes by increasing the measured error rate when compared to their training period (see Fig. 1). To address such a challenge, the network operator must conduct periodic model updates, which usually require weeks or even months to be fulfilled. As a result, designed classification schemes must be able to keep their reliability for more extended periods while an updated model is still being designed.

Our proposed model aims to solve this problem by implementing the classification task as a classification with a reject option. Our proposed scheme uses the rejector to identify two types of incorrect predictions, namely *ambiguity* and *novelty*. On the one hand, *ambiguity* refers to decisions that the predictor cannot perform with high reliability, e.g., because they are closer to the predictor's decision boundary. On the other hand, *novelty* handles decisions that are highly different from those experienced during model training, e.g., because they are a new kind of behavior. Fig. 3 shows the classification pipeline with the added rejector module. It follows a traditional network-based intrusion detection pipeline (see Sec. 2.1) with an added rejector module implemented before the alert module.

A ML model with a reject option combines the used predictor w with a rejector r to implement the rejection decision function on a given input event. Given a model w that outputs a pair of classification confidence values $\alpha = \{\alpha_{normal}, \alpha_{attack}\}$ for every input event x. Where α measures the event confidence to be either *normal* or *attack* classes, respectively, such that $\alpha \in \mathbb{R}[0,1]$. The rejector's r goal is to reject or accept the classification based on its associated confidence values α, according to the following function:

$$w(x) \begin{cases} \emptyset & \text{if } r(\alpha) = reject \\ w(x) & \text{otherwise} \end{cases} \tag{5}$$

where $\emptyset$ marks input events likely to be incorrect decisions the predictor performs. As part of the problem, we need to identify and adequately address misclassifications performed by the predictor due to the evolving behavior of network traffic. We assess the predictor's classification confidence values α to implement the rejector in a classifier agnostic rationale to achieve such a goal. Equation (7) shows the rejector implementation function in our proposal.

$$r(\alpha, t) \begin{cases} \emptyset & \text{if } \alpha \le t \\ \alpha & \text{otherwise} \end{cases} \tag{6}$$

$$d(\alpha, t) \begin{cases} \overbrace{r(\alpha_{normal}, t_{normal})}^{\text{Normal Classified}} & \text{if } \alpha_{normal} > \alpha_{attack} \\ \underbrace{r(\alpha_{attack}, t_{attack})}_{\text{Attack Classified}} & \text{otherwise} \end{cases} \tag{7}$$

where t denotes the pair of acceptance threshold values for each class such that $t \in \mathbb{R}[0,1]$, and d a decision function coped with the proposal ejector module.

Thus, the input events acceptance is established according to their classification confidence level α evaluated against an acceptance threshold t. Low-confident classifications are more likely to be related to *ambiguity* or *novelty* events, which are assumed to occur due to the evolving behavior of network traffic. Therefore, low-confident classifications are rejected by our system and adequately stored for later model updates. In contrast, high-confident classifications are assumed to be network events known to the underlying deployed ML model, thus, can be reliably accepted by our system. Therefore, our proposal solves the following equation to implement the rejector's threshold optimization.

$$T(w, \mathcal{D}, y) = \underset{t^{\times 2} \in \mathbb{R}[0,1]}{\arg\min}\, \beta E(w, \mathcal{D}, y, t) + \gamma R(w, \mathcal{D}, y, t) \tag{8}$$

where $T(w, \mathcal{D}, y)$ is an acceptance threshold search function for a ML model w, on a given dataset $\mathcal{D}$, with an associated label y. The function goal is to find an acceptance threshold t that minimizes the sum of the functions $E(w, \mathcal{D}, y, t)$ and $R(w, \mathcal{D}, y, t)$, where E measures the error rate on the accepted events using threshold t, and R measures the rejection rate with the used threshold. The error and rejection rates are multiplied by previously defined β and γ values based on the network operator's needs. A higher rejector threshold can improve the system's reliability while increasing the number of rejected events as a trade. In contrast, a lower rejector threshold decreases the number of rejected events while affecting the system's reliability as time passes. Our insight is to identify network events unknown to the deployed ML model and update it based on the newly identified behavior. As a result, we can significantly decrease the computational costs during model updates and reduce the number of network events that require labeling.

The proposal classification pipeline employs a traditional ML-based NIDS deployment on the FL peers (Fig. 3, *Reliable Classification*). In such a context, the network events are collected through a *Data Acquistion* module, while the behavior of the collected events is extracted by a *Feature Extraction* module. A *Classification* module uses the built feature vector as input, which applies the ML model and outputs an associated classification confidence value. The classification confidence value is used as input by our *Rejector* module, which evaluates if a given classification confidence threshold is met according to Eq. (7). Highly confident classifications are accepted by our model and can be used to trigger alerts. In contrast, low confident classifications are rejected by our model and stored in a dataset $\mathcal{D}_i^{\emptyset}$ for latter model updates within each FL peer (see Sec. 5.2).

Algorithm 1 Reliable Classification Pipeline.

```
Require:
  Model $w_G$
  Event $x = \{f_1, f_2, ..., f_N\}$
  Threshold $t = \{t_{attack}, t_{normal}\}$
  procedure CLASSIFICATION($w_G$, $x$, $t$)
    $\alpha \leftarrow classify(w_G, x)$
    if $\alpha_{normal} > \alpha_{attack}$ and $\alpha_{normal} \ge t_{normal}$ then
      $alert(x, normal)$
    else if $\alpha_{attack} \ge t_{attack}$ then
      $alert(x, attack)$
    else
      $reject(x)$
    end if
  end procedure
```

Algorithm 1 shows our reliable classification pipeline procedure (Fig. 3, *Reliable Classification*). It receives as input a ML model w, an instance x represented by a feature set f, and an acceptance threshold t associated with each class (*normal* or *attack*). The model classifies the input instance, outputting a pair of classification confidence value α. According to the classification confidence, the event triggers alerts if it surpasses a previously defined rejector threshold. Otherwise, the event is rejected and used by our model update pipeline (see Sec. 5.2).

As a result, our proposed classification pipeline can ensure the system's reliability over time, even in the presence of an outdated ML model. Notwithstanding, our proposed scheme can also proactively identify new kinds of network traffic behavior that should be used for model updates. This is because our model will reject new network traffic in an unsupervised fashion due to the low classification confidence.

5.2. FL-based model update

The model update is a challenging task in ML-based NIDSs. This is because model updates require the provision of an updated labeled dataset and the execution of a computationally expensive model training process. On the one hand, building a new training dataset usually requires the collection of new network traffic behavior and human assistance for adequate labeling. On the other hand, model training is a usually computationally expensive process.

Our proposed scheme implements periodic model updates in a three-phase process to address such a challenge. First, model updates are conducted in a FL rationale, improving the quality of the built model while enabling network operators to leverage the participant organizations CTI. Second, model updates are performed using the system's previously rejected events (see Sec. 5.1). Consequently, we can significantly reduce the number of events that must be used for model updates. Our main assumption is that model updates can be performed only using the events that the ML model could not classify reliably, fine-tuning the system as time passes. Third, model updates are conducted following a transfer learning approach, using the outdated model to reduce the computational costs and leverage the previously trained knowledge available on the model.

Our proposed FL-based model update pipeline is shown in Fig. 3. It considers a central server s and a set of k Peers P_i. The centralized server is assumed to be benign and is responsible for conducting the tasks of *Initialization*, *Distribution*, and *Aggregation*. Each peer P_i is responsible for conducting the *Local Training* task, and holds an *Unreliable Dataset* $D_i^{\emptyset}$ that stores the previously peer rejected events $\emptyset$ (see Eq. (7)) since the last execution of the model update. Each peer P_i is responsible to conduct the *Unreliable Dataset* labeling to compound a *Labeled Dataset* $\mathcal{D}_i$. The dataset labeling can be conducted by automated *misuse-based* intrusion detection tools or through network operator assistance.

Algorithm 2 Reliable Peer Update.

```
Require:
  Model $w_G^t$                                   ▷ Global Round Model
  Unreliable Dataset $\mathcal{D}_i^\emptyset$    ▷ Peer $i$ rejected events
  Label Provider $l$                              ▷ Network Operator
  procedure PeerUpdate($w_G^t$, $\mathcal{D}_i^\emptyset$, $l$)
    $\mathcal{D}_i \leftarrow label(\mathcal{D}_i^\emptyset, l)$    ▷ Label rejected dataset
    $w_i^t \leftarrow update(w_G^t, \mathcal{D}_i)$                 ▷ Update model
    $export(w_i^t)$                                                  ▷ Export to central server
  end procedure
```

The model update task is periodically executed, e.g. every month by the central server s (Fig. 3, *Global Update Trigger*). The procedure initially distributes the global round model w_G^t to all FL peers. Without sharing their private data, each peer P_i updates their local model according to their previously rejected and labeled events $\mathcal{D}_i$. The labeled dataset $\mathcal{D}_i$ is used to update the w_G^t model to compound a local model w_i^t following a transfer learning rationale, fine-tuning the model to the events that were previously rejected by the peer reliable classification pipeline. The updated local model w_i^t is returned to the central server, which performs the model aggregation task, outputting a global model w_G^{t+1}. The model update procedure is repeated until T execution rounds are reached. Finally, the built global model w_G is sent back to all FL peers, finalizing the model update task.

Algorithm 2 shows the model update procedure. It is executed on every participant peer, on every global round model update execution, it receives as input a model w_G^t, an unreliable dataset $\mathcal{D}_i^\emptyset$ containing the peer i rejected events, and a label provider l. Recalling that the label provider can be conducted using automated *misuse-based* intrusion detection tools or network operator assistance. The label provider l labels the rejected dataset $\mathcal{D}_i^\emptyset$ to compound a labeled dataset $\mathcal{D}_i$. The labeled dataset $\mathcal{D}_i$ updates the received model w_G^t resulting in a local model w_i^t. Finally, the peer-updated classifier w_i^t is exported to the central server to aggregate a global model counterpart.

As a result, the benefits of our FL-based model update task are twofold. First, participants do not need to share their private data while still being able to use other participants' knowledge, as represented by their local models. Therefore, several participants can collaborate during the model update task without involving a huge amount of training data. Second, the model update procedure is based on the rejected events over time. Thus, the training data used throughout updates significantly decreases while the model is fine-tuned according to the previously rejected events. Such a characteristic enables operators to fine-tune their deployed ML model based on the newly network traffic.

6. Evaluation

The proposal evaluation aims at answering the following **RQ**:

- **(RQ3)** *How does the traditional FL perform when facing new network traffic?*
- **(RQ4)** *How does the proposed rejector technique assist in improving the classification reliability?*
- **(RQ5)** *How does our proposed model perform over time with periodic model updates?*

The following subsections further describe the proposed model-building procedure and its evaluation.

6.1. Model building

The proposed model is evaluated by using a Multilayer Perceptron (MLP) classifier implemented with 500 hidden neurons, a *relu* activation function, and *adam* optimization. The parameters are defined empirically. The classifier is implemented through *scikit-learn* API v0.24 (scikit-learn, 2023). We adopt the MLP for the FL implementation, aggregating the local models into a single global counterpart. Thus, we implement the global model aggregation by computing the average of local MLP weights by leveraging the FedAVG algorithm.

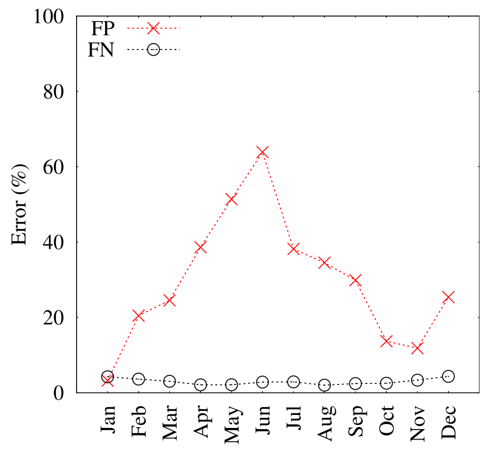


Fig. 4. Classification performance of the traditional FL implementation without model updates over time. System is only trained with January data.

Following, we describe the steps of our implemented FL-based training and model update procedure (see Sec. 2.3 and Fig. 3):

1) *Initialization.* If it is proposal initialization, the central server s randomly initializes a global round model w_G^t. Conversely, if a model update is conducted, the previously trained global model w_G is used.
2) *Distribution.* Global round model w_G^t is sent to all FL peers.
3) *Local Training.* Each peer executes Algorithm 2 and updates the received model w_G^t to compound a model w_i^t. The model is fine-tuned using 500 epochs. If it is the initial training, each peer i updates the received model w_G^t with randomly selected events from the training dataset. Conversely, if a model update is conducted, training is performed based on the peer i rejected events stored on the dataset $\mathcal{D}_i$.
4) *Aggregation.* The built model w_i^t is sent to central server s. The central server executes the FedAVG algorithm to compound a global model w_G^{t+1}. If $t \leq 10$, another round of the model training or update is executed (phase 2 to 4). Otherwise, model training is finalized, and the built global model w_G is sent to peers for intrusion detection purposes.

We consider a proposal deployment scenario with 10 FL peers (Fig. 3, Peers). We adopt a random undersampling without replacement to balance the occurrence between the classes during the initial training procedure.

6.2. Reliable federated learning

Our first experiment aims at answering **RQ3** and investigates the accuracy performance of the traditional FL. We execute the training procedure by considering a 10 peers scenario implemented through the FedAVG without using our proposed scheme. We perform the traditional FL by randomly splitting, in a stratified manner, the January data throughout peers. Each peer builds their local model while the global model is built by the central server based on each peer's local model (see Sec. 6.1).

Fig. 4 shows the classification accuracy of the traditional FL scheme without periodic model updates. Compared with the traditional techniques (see Fig. 1), we notice a decrease in the FL model accuracy when we do not perform model updates as time passes. For example, the non-updated traditional FL scheme reaches a FP rate of 64% in June, while the traditional kNN reaches 58% (Fig. 4 *vs.* 1b). Similarly, the accuracy decrease is experienced a month after the training phase, increasing the

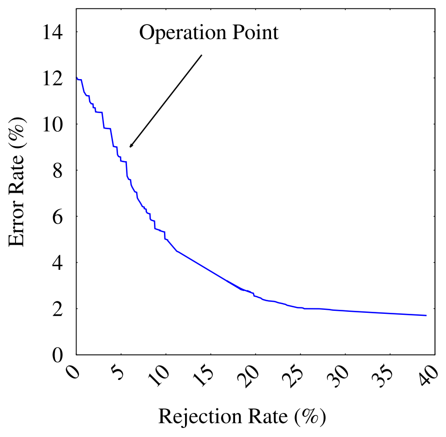


Fig. 5. Error-reject tradeoff of our proposed classification verification technique. Classifier is trained on January data and evaluated in February.

FP rate in 22%. As a result, the traditional FL cannot keep its reliability as time passes, demanding the execution of periodic model updates.

Our second experiment aims to answer **RQ4**, and it evaluates how our proposed verification technique can improve the system's classification accuracy even without performing the model update procedure. To achieve such a goal, we implement our proposed rejector's threshold optimization (see Eq. (8)) considering a variation on threshold t in a 0.01 interval. We compute the MLP classifier confidence value α through the *predict proba* function from *scikit-learn* API (Eq. (7), α). As the rejector operation point must be defined according to the network operator's needs, we implement the optimization considering $\beta = 1.0$ and $\gamma = 1.0$. Hence, rejection and error account for the same weight during the threshold-finding process. We first evaluate the error-reject tradeoff in February using the January-trained model (shown in Fig. 4).

Fig. 5 shows the error *vs.* rejection rate tradeoff in February when we adopt our proposed rejection technique using our implemented threshold finding approach (Eq. (8)). It is possible to note that the proposed verification technique can improve detection accuracy by assessing the classification confidence values. For instance, the network operator can decrease the system's error rate by 10% if a 20% rejection rate can be tolerated. Notwithstanding, a rejection of only 5% can decrease the system's error rate by 4%. As a result, our proposed verification technique can be used to guarantee the system's reliability as time passes, even when model updates are not conducted.

Leveraging the verification technique, we evaluate the detection accuracy of our model if we do not assume the involvement of model updates. Indeed, we use the verification technique to perform such a task and suppress classifications with low confidence values during the accuracy computation. To achieve such a goal, we consider a rejection operation point of 5%, which uses an α_{normal} of 0.92 and α_{attack} of 0.87 (Algorithm 1). It is important to note that the rejection operation point must be defined according to the network operator's needs. On the one hand, we can improve detection accuracy by rejecting more instances. On the other hand, we can classify additional instances if we tolerate a certain degree of error rate. Our evaluation considers a realistic scenario wherein the network operator tolerates only 5% of rejected events.

Fig. 6 shows the classification accuracy of our proposal without periodic model updates, coped with the rejector module using the previously selected acceptance thresholds. Our proposed model can improve detection accuracy when assessing the input event classification reliability. In detail, the verification technique can enhance the average FP rate in 12.9%, while only rejecting an average of 13.5% of instances when compared to its no-update counterpart (Fig. 4 *vs.* 6). The proposed verifier module can keep the system reliable for extended intervals even when no model updates are not considered.

To answer **RQ5**, we further investigate how the execution of periodic model updates based on rejected events can enhance the system's

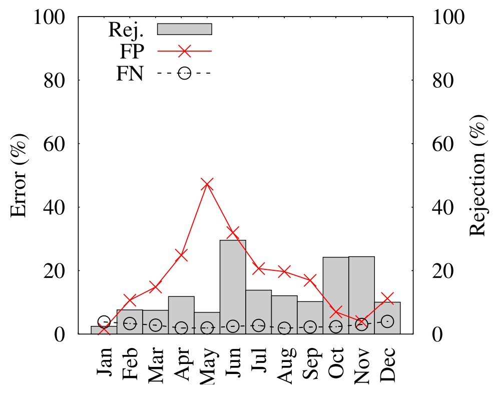


Fig. 6. Proposed model without updates using verification.

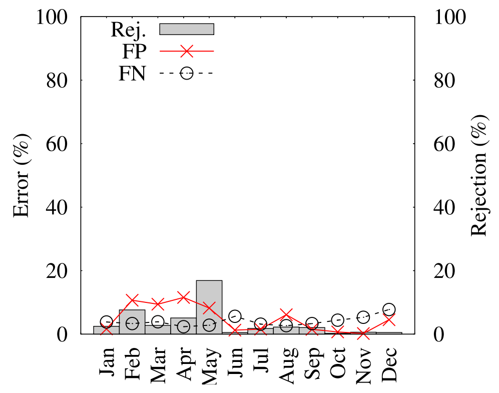


Fig. 7. Proposed model with monthly updates and verification.

accuracy (Algorithm 2). To this aim, we perform monthly FL-based model updates according to the events rejected by each peer (Fig. 3, *Unreliable Dataset* ($D_i^{\emptyset}$)). At the same time, the central server s performs the aggregation task as depicted by the FedAVG algorithm (see Sec. 6.1). Similarly, model updates are executed monthly, as evaluated with the traditional approaches.

Fig. 7 shows the classification accuracy of our model with periodic monthly model updates. Our proposed scheme significantly improves detection accuracy while decreasing the number of rejected events. More specifically, when compared to its no-update counterpart (Fig. 6 *vs.* 7), the proposed scheme can provide an average FP rate of only 3.3% while rejecting only 3.6% of instances, i.e., a 9.9% rejection rate reduction. As a result, the model update task based on the rejected instances could fine-tune the model as time passed. Notwithstanding, by performing model updates through a FL rationale, organizations can execute such a task more frequently, given that the participants will collect more network traffic and share their CTI.

Moreover, we investigate how our scheme performs when compared to traditional techniques. Fig. 8 shows the F-Measure throughout the time of our scheme vs. the traditional approaches. Our proposed model improves classification accuracy with and without considering the model updates. On average, in a no-update setting (Fig. 8a), our scheme provides an F-Measure of 0.89, an increase of 0.23 and 0.13 when compared to the RF and kNN, respectively. Assuming to use the model updates (Fig. 8b), we reach an average F-Measure of 0.96, improving it by 0.03 and 0.04 when compared to the RF and kNN, respectively.

The main advantage of our scheme concerns the amount of data that must be labeled to conduct model updates as time passes. Fig. 9a shows the cumulative number of network events that must be labeled as time passes for each selected technique. Our proposal required an average of only 6.5% of events when compared to the traditional approach (Proposal FL vs. Traditional FL). Considering the events required after the initial training in January, our proposed scheme significantly im-

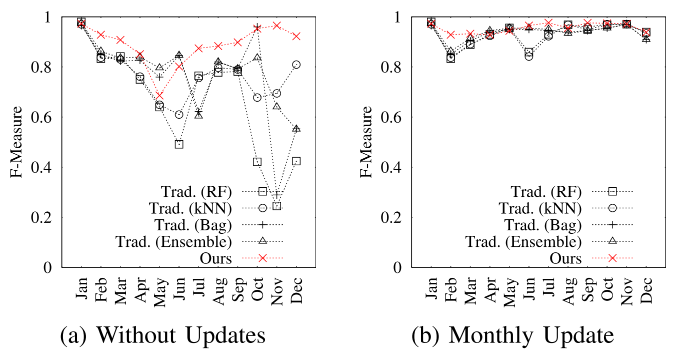


Fig. 8. Accuracy behavior of selected techniques on *MAWIFlow* dataset.

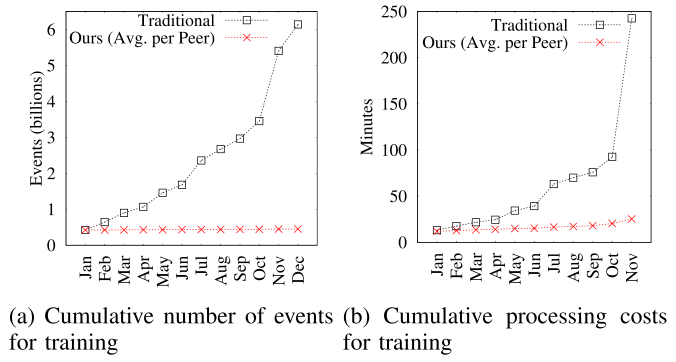


Fig. 9. Proposed model operational comparison with traditional techniques in terms of used events and processing costs during training phase.

proves the traditional approach, demanding an average of only 0.3% of events to be labeled as time passes. Notwithstanding, as organizations can share their CTI due to the application of FL, we can further decrease the needed training data according to the number of available peers.

We further investigate the training processing costs of our solution when compared to traditional techniques. To achieve such a goal, we measure the training time as time passes used by our proposal with the verifier and monthly updates (Fig. 7) *vs.* the monthly updated Ensemble approach (Fig. 2d). The experiments were executed in a 10-node cluster, each equipped with an 8-core Intel i7 CPU, 16GB of memory, interconnected through a gigabit network, running on top of an Ubuntu v.22.04 OS. The traditional Ensemble technique uses the entire node processing capabilities, while our proposed model uses the entire cluster processing when available. Fig. 9b shows the cumulative training processing costs of our solution when compared to the Ensemble monthly updated technique (Fig. 2d). Our proposed scheme significantly decreases processing costs when compared to the traditional technique. Our proposal requires in average only 10% of processing costs per node compared to the traditional Ensemble approach. This is because our proposal can proactively select which events should be used for model updates (Fig. 9a), significantly easing the processing costs for model updates as time passes.

Finally, we investigate how our proposed scheme can relax the frequency with which model updates are conducted. We vary the frequency of conducting the model update on the rejected instances to achieve such a goal. Fig. 10 shows the classification performance according to the model update periodicity in our proposal. Our proposal is not significantly affected by a longer model lifespan, with a marginal impact on the system's accuracy. For instance, the monthly updated model provides an average F-Measure of 0.96, while its counterpart, updated every semester, reaches 0.90, an impact of only 0.06. The longer model lifespan significantly benefits a realistic operational deployment of ML-based NIDS. This is because the network operator can use their

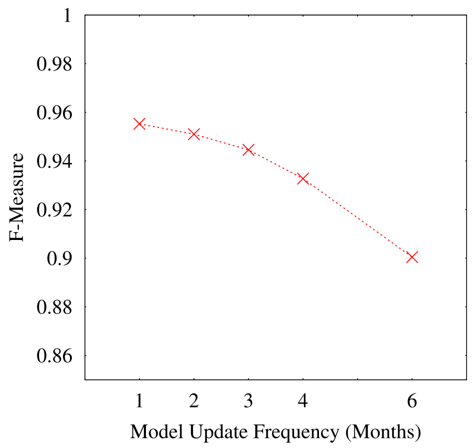


Fig. 10. Proposed model performance according to model lifespan, i.e. frequency of model updates.

built ML model for longer periods before a model update is conducted, and when made so, fewer events must be provided and fewer processing costs will be required.

7. Conclusion

Over the past decades, several works have proposed highly accurate ML-based techniques for NIDSs, yet, their actual deployment on production environments is hardly observed. This paper has shown that the non-stationary behavior of network traffic makes ML-based NIDS unreliable months after the training phase, demanding unfeasible model updates to be periodically conducted. To address such a shortcoming, we proposed a FL model aiming the reliability of NIDS. Even in the presence of new network traffic behavior, classification reliability is ensured through classification with a reject option rationale. Notwithstanding, model updates are conducted through FL-based training on the rejected events over time. As a result, organizations can share their network traffic knowledge extracted from their private data without affecting data privacy.

CRediT authorship contribution statement

Roger R. dos Santos: Conceptualization, Methodology, Software, Writing – original draft. **Eduardo K. Viegas:** Data curation, Software, Supervision, Validation, Writing – original draft. **Altair O. Santin:** Investigation, Supervision, Writing – review & editing. **Pietro Tedeschi:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The dataset used in the experiments conducted throughout this paper is publicly available at https://secplab.ppgia.pucpr.br/reliablefedlearningids.

Acknowledgement

This work was partially sponsored by Brazilian National Council for Scientific and Technological Development (CNPq) grant n. 304990/2021-3.

References

Al-Hadhrami, Y., Hussain, F.K., 2020. Real time dataset generation framework for intrusion detection systems in IoT. Future Gener. Comput. Syst. 108, 414–423.

Alghamdi, R., Bellaiche, M., 2023. A cascaded federated deep learning based framework for detecting wormhole attacks in iot networks. Comput. Secur. 125, 103014.

Arp, Daniel, Quiring, Erwin, Pendlebury, Feargus, Warnecke, Alexander, Pierazzi, F., Wressnegger, Christian, Cavallaro, L., Rieck, Konrad, 2021. Dos and Don'ts of Machine Learning in Computer Security, 2022nd ed. Usenix Security Symposium (USENIX). USENIX.

Campos, E.M., Saura, P.F., González-Vidal, A., Hernández-Ramos, J.L., Bernabé, J.B., Baldini, G., Skarmeta, A., 2022. Evaluating federated learning for intrusion detection in Internet of things: review and challenges. Comput. Netw. 203, 108661.

Dromard, J., Roudiere, G., Owezarski, P., 2017. Online and scalable unsupervised network anomaly detection method. IEEE Trans. Netw. Serv. Manag. 14 (1), 34–47.

Fontugne, R., Borgnat, P., Abry, P., Fukuda, K., 2010. MAWILab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In: Proc. of the 6th Int. Conf. on Emerging Networking EXperiments and Technologies (CoNEXT).

Gates, C., Taylor, C., 2006. Challenging the anomaly detection paradigm: a provocative discussion. In: Proc. of the Workshop on New Security Paradigms (NSPW), pp. 21–29.

Hei, X., Yin, X., Wang, Y., Ren, J., Zhu, L., 2020. A trusted feature aggregator federated learning for distributed malicious attack detection. Comput. Secur. 99, 102033.

Huang, Y., Ma, M., 2023. ILL-IDS: an incremental lifetime learning IDS for VANETs. Comput. Secur. 124, 102992.

Jiang, J., Liu, F., Ng, W.W.Y., Tang, Q., Wang, W., Pham, Q.-V., 2022. Dynamic incremental ensemble fuzzy classifier for data streams in green Internet of things. IEEE Trans. Green Commun. Netw. 6 (3), 1316–1329.

Kilincer, I.F., Ertam, F., Sengur, A., 2021. Machine learning methods for cyber security intrusion detection: datasets and comparative study. Comput. Netw. 188, 107840.

Lee, J.-S., Chen, Y.-C., Chew, C.-J., Chen, C.-L., Huynh, T.-N., Kuo, C.-W., 2022. Conn-ids: intrusion detection system based on collaborative neural networks and agile training. Comput. Secur. 122, 102908.

Li, B., Wang, Y., Xu, K., Cheng, L., Qin, Z., 2022. DFAID: density-aware and feature-deviated active intrusion detection over network traffic streams. Comput. Secur. 118, 102719.

Li, X., Hu, Z., Xu, M., Wang, Y., Ma, J., 2021. Transfer learning based intrusion detection scheme for Internet of vehicles. Inf. Sci. 547, 119–135.

Mahdavi, E., Fanian, A., Mirzaei, A., Taghiyarrenani, Z., 2022. ITL-IDS: incremental transfer learning for intrusion detection systems. Knowl.-Based Syst. 253, 109542.

Maseer, Z.K., Yusof, R., Bahaman, N., Mostafa, S.A., Foozy, C.F.M., 2021. Benchmarking of machine learning for anomaly based intrusion detection systems in the CICIDS2017 dataset. IEEE Access 9, 22351–22370.

MAWI, 2023. MAWI working group traffic archive - samplepoint F. Available https://mawi.wide.ad.jp/mawi/ [Online].

Mills, R., Marnerides, A.K., Broadbent, M., Race, N., 2022. Practical intrusion detection of emerging threats. IEEE Trans. Netw. Serv. Manag. 19 (1), 582–600.

Molina-Coronado, B., Mori, U., Mendiburu, A., Miguel-Alonso, J., 2020. Survey of network intrusion detection methods from the perspective of the knowledge discovery in databases process. IEEE Trans. Netw. Serv. Manag. 17 (4), 2451–2479.

Mothukuri, V., Khare, P., Parizi, R.M., Pouriyeh, S., Dehghantanha, A., Srivastava, G., 2022. Federated-learning-based anomaly detection for IoT security attacks. IEEE Int. Things J. 9 (4), 2545–2554.

Papadogiannaki, E., Ioannidis, S., 2021. A survey on encrypted network traffic analysis applications, techniques, and countermeasures. ACM Comput. Surv. 54 (6).

Ramkumar, M., Reddy, P.B., Thirukrishna, J., Vidyadhari, C., 2022. Intrusion detection in big data using hybrid feature fusion and optimization enabled deep learning based on spark architecture. Comput. Secur. 116, 102668.

Saba, T., Rehman, A., Sadad, T., Kolivand, H., Bahaj, S.A., 2022. Anomaly-based intrusion detection system for iot networks through deep learning model. Comput. Electr. Eng. 99, 107810.

scikit-learn, 2023. Machine Learning in Python. Available https://scikit-learn.org/ [Online].

Sommer, R., Paxson, V., 2010. Outside the closed world: on using machine learning for network intrusion detection. In: 2010 IEEE Symposium on Security and Privacy. IEEE.

Statistics, 2023. Kaspersky Security Bulletin 2022. Available https://securelist.com/ksb-2022-statistics/108129/ [Online].

Sun, Y., Ochiai, H., Esaki, H., 2020. Intrusion detection with segmented federated learning for large-scale multiple LANs. In: 2020 International Joint Conference on Neural Networks (IJCNN). IEEE.

Thakkar, A., Lohiya, R., 2021. A survey on intrusion detection system: feature selection, model, performance measures, application perspective, challenges, and future research directions. Artif. Intell. Rev. 55 (1), 453–563.

Viegas, E., Santin, A., Bessani, A., Neves, N., 2019. BigFlow: real-time and reliable anomaly-based intrusion detection for high-speed networks. Future Gener. Comput. Syst. 93, 473–485.

Wahab, O.A., 2022. Intrusion detection in the IoT under data and concept drifts: online deep learning approach. IEEE Int. Things J. 9 (20), 19706–19716.

Wu, Z., Gao, P., Cui, L., Chen, J., 2022. An incremental learning method based on dynamic ensemble RVM for intrusion detection. IEEE Trans. Netw. Serv. Manag. 19 (1), 671–685.

Yamin, M.M., Katt, B., Gkioulos, V., 2020. Cyber ranges and security testbeds: scenarios, functions, tools and architecture. Comput. Secur. 88, 101636.

Yang, Z., Liu, X., Li, T., Wu, D., Wang, J., Zhao, Y., Han, H., 2022. A systematic literature review of methods and datasets for anomaly-based network intrusion detection. Comput. Secur. 116, 102675.

Yuan, S., Li, H., Zhang, R., Hao, M., Li, Y., Lu, R., 2021. Towards lightweight and efficient distributed intrusion detection framework. In: 2021 IEEE Global Communications Conference (GLOBECOM). IEEE.

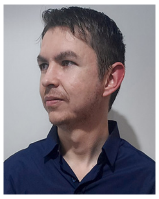
Zeng, Z., Peng, W., Zeng, D., 2022. Improving the stability of intrusion detection with causal deep learning. IEEE Trans. Netw. Serv. Manag. 19 (4), 4750–4763.

Zhang, C., Jia, D., Wang, L., Wang, W., Liu, F., Yang, A., 2022. Comparative research on network intrusion detection methods based on machine learning. Comput. Secur. 121, 102861.

Zhao, Q., Chen, M., Gu, Z., Luan, S., Zeng, H., Chakrabory, S., 2022. CAN bus intrusion detection based on auxiliary classifier GAN and out-of-distribution detection. ACM Trans. Embed. Comput. Syst. 21 (4), 1–30.

Zhou, Y., Ye, Q., Lv, J., 2021. Communication-efficient federated learning with compensated overlap-fedavg. IEEE Trans. Parallel Distrib. Syst. 33 (1), 192–205.

Zoppi, T., Ceccarelli, A., Puccetti, T., Bondavalli, A., 2023. Which algorithm can detect unknown attacks? Comparison of supervised, unsupervised and meta-learning algorithms for intrusion detection. Comput. Secur. 127, 103107.



Roger R. dos Santos Received the BS degree in information systems from PUCPR in 2014, the MSC degree in computer science in 2020, and is currently working toward the PhD degree at PUCPR. His research interests include machine learning and computer security.

Eduardo K. Viegas Received the BS degree in computer science in 2013, the MSC degree in computer science in 2016 from PUCPR, and the PhD degree from PUCPR in 2018. His research interests include machine learning, network analytics and computer security.

Altair O. Santin Received the BS degree in Computer Engineering from the PUCPR in 1992, the MSc degree from UTFPR in 1996, and the PhD degree from UFSC in 2004. He is a full professor of Graduate Program in Computer Science (PPGIa) and head of Security & Privacy Lab (SecPLab) at PUCPR. He is a member of the IEEE, ACM, and the Brazilian Computer Society.

Pietro Tedeschi is currently Senior Security Researcher at Technology Innovation Institute, Autonomous Robotics Research Center, Abu Dhabi, United Arab Emirates, from January 2022. He obtained his Ph.D. in Computer Science and Engineering from Hamad Bin Khalifa University (HBKU), College of Science and Engineering (CSE), Doha, Qatar in December 2021. He received his Master's degree with honors in Computer Engineering at Politecnico di Bari, Italy. From 2017 to 2018, he worked as Security Researcher at CNIT (Consorzio Nazionale Interuniversitario per le Telecomunicazioni), Italy, for the EU H2020 SymbIoTe project. He is also serving in the TPC of several conferences. His major research interests include security and privacy in Unmanned Aerial Vehicles, Wireless, Internet of Things, Cyber-physical Systems, Applied Cryptography.