Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2021.DOI

An end-to-end framework for machine learning-based network intrusion detection system

GUSTAVO DE CARVALHO BERTOLI[1], LOURENÇO ALVES PEREIRA JUNIOR[2], OSAMU SAOTOME[3], ALDRI L. SANTOS[4], FILIPE ALVES NETO VERRI[2], CESAR AUGUSTO CAVALHEIRO MARCONDES[2], SIDNEI BARBIERI[5], MOISES S. RODRIGUES[5], AND JOSÉ M. PARENTE OLIVEIRA[2]

[1]PhD Candidate, Aeronautics Institute of Technology (ITA), São José dos Campos 12228-900, Brazil
[2]Computer Science Division, Aeronautics Institute of Technology (ITA), São José dos Campos 12228-900, Brazil
[3]Electronics Engineering Division, Aeronautics Institute of Technology (ITA), São José dos Campos 12228-900, Brazil
[4]Center for Computational Security Science (CCSC), Computer Science Department, Federal University of Minas Gerais (UFMG), 31270-901, Brazil
[5]Cyber Defense Command, Brazilian Army, Brazil

Corresponding author: Lourenço Alves Pereira Júnior (e-mail: ljr@ita.br)

This work was supported in part by ITA's Programa de Pós-graduação em Aplicações Operacionais (ITA/PPGAO).

ABSTRACT The increase of connected devices and the constantly evolving methods and techniques by attackers pose a challenge for network intrusion detection systems from conception to operation. As a result, we see a constant adoption of machine learning algorithms for network intrusion detection systems. However, the dataset used by these studies has become obsolete regarding both background and attack traffic. This work describes the AB-TRAP framework that enables the use of updated network traffic and considers operational concerns to enable the complete deployment of the solution. AB-TRAP is a five-step framework consisting of (i) the generation of the attack dataset, (ii) the bonafide dataset, (iii) training of machine learning models, (iv) realization (implementation) of the models, and (v) the performance evaluation of the realized model after deployment. We exercised the AB-TRAP for local (LAN) and global (internet) environments to detect TCP port scanning attacks. The LAN study case presented an f1-score of 0.96, and an area under the ROC curve of 0.99 using a decision tree with minimal CPU and RAM usage on kernel-space. For the internet case with eight machine learning algorithms with an average f1-score of 0.95, an average area under the ROC curve of 0.98, and an average overhead of 1.4% CPU and 3.6% RAM on user-space in a single-board computer. This framework has the following paramount characteristics: it is reproducible, uses the most up-to-date network traffic, attacks, and addresses the concerns to the model's realization and deployment.

INDEX TERMS cybersecurity, datasets, intrusion detection system, machine learning, network security, supervised learning

NOMENCLATURE

AUC	Area Under the Curve.
$B5G$	5G and beyond.
$CoAP$	Constrained Application Protocol.
CPS	Cyber-Physical System.
CSV	Comma-separated values.
$DDoS$	Distributed Denial of Service.
DL	Deep Learning.
$DNP3$	Distributed Network Protocol 3.
DoS	Denial of Service.
DT	Decision tree.
$eBPF$	Extended Berkeley Packet Filter.
$FPGA$	Field Programmable Gate Array.
FPR	False Positive Rate.
FTP	File Transfer Protocol.
$HIDS$	Host Intrusion Detection System.
$HTTP$	Hypertext Transfer Protocol.
ICS	Industrial Control System.
IDS	Intrusion Detection System.
IEC	International Electrotechnical Commission.

IoT	Internet of Things.
IP	Internet Protocol.
IT	Information technology.
kNN	k-Nearest Neighbours.
LAN	Local Area network.
LKM	Linux Kernel Module.
LR	Logistic Regression.
$MitM$	Man in the Middle.
ML	Machine Learning.
MLP	Multi-Layer Perceptron.
$MQTT$	Message Queuing Telemetry Transport.
NB	Naïve Bayes.
NF	Netfilter.
NIC	Network Interface Card.
$NIDS$	Network Intrusion Detection System.
OTA	Over-The-Air.
$pcap$	Packet capture.
SQL	Structured Query Language.
RF	Random Forest.
ROC	Receiver Operating Characteristic.
SBC	Single Board Computer.
SDN	Software Defined Network.
SSH	Secure Shell.
SVM	Support Vector Machines.
TCP	Transmission Control Protocol.
TPR	True Positive Rate.
TTL	Time-To-Live.
UDP	User Datagram Protocol.
VM	Virtual Machine.
VPN	Virtual Private Network.
XDP	Express Data Path.
XGB	Extreme Gradient Boosting (XGBoost).

I. INTRODUCTION

NOWADAYS the internet is ubiquitous and an enabler of the concept known as the internet of things (IoT). This concept allows billions of devices to be accessible through the internet, revolutionizing our day-to-day lives. In this reality, and considering the deployment of B5G (5G and beyond) technology that reduces communications latency, disruptive applications will become true, such as autonomous driving, autonomous traffic management, and remote surgery. Therefore, we must expect a considerable change in the network traffic pattern and new attacks targeting these applications. In this context, cybersecurity poses a significant technical challenge since the lack of base characteristics (i.e., availability, confidentiality, and integrity) hindermost of the envisioned vertical applications [1].

The interconnected nature of the devices requires the security-by-design approach, which brings pros and cons to building security protection. On the IoT domain, there is the use of cryptography, secure hardware (to avoid tampering), and intrusion detection systems (IDS) [2]. Also, there are other mechanisms envisioned for IoT such as blockchain, fog computing, machine learning (ML), and edge computing [3]. All of them have to deal with the inherent property of resource and energy constraints seen in IoT devices. Cyber-physical systems (CPS) comprise industrial control systems (ICS), smart grids, smart cars, and medical devices. However, the security flaws in that system are recurrent and current protection may not be adequate as new exploitation techniques arise. Moreover, a big challenge in this domain is that security incidents can lead to a catastrophic scenario (directly affecting people), the applicable security mechanisms in this domain are those also applicable for IoT, and the CPS also introduces requirements for access control [4].

The security of communications imposes a constantly evolving characteristic due to attacks and network infrastructure. As new vulnerability disclosure introduces unknown attacks, attempts in the network [5], and by the changing of network infrastructure caused by mobile networks, or by fresh devices entering the network like those from IoT and CPS. This inherent characteristic of networked communications along with the massive amount of available data (or the capability to obtain it) take the use of ML that, in summary, aims to learn from data. Therefore, this application field can also address this evolving characteristic. For instance, ML models can deal with estimable features from transport and network internet protocols, like UDP, TCP, and IP, and as a result addressing upper layers protocols from IoT and CPS domain such as MQTT, CoAP, Modbus, DNP3, IEC 61850, and those from IT such as HTTP, FTP, Telnet, SSH, among others.

Consider the Cyber-kill-chain attack model; the reconnaissance phase is responsible for revealing the potential target hosts. Thereby building protection modules to prevent devices from being discovered by malicious agents indicated a promising approach. It acts as an invisibility shield to devices, and the task comprises identifying abnormal patterns and avoiding responding to them. However, the main barrier of this approach relates to the lack of appropriate datasets [6]; in other words, easy to generate and up-to-date with recent attack samples datasets are seldom available.

In this paper, we present AB-TRAP (**A**ttack, **B**onafide, **T**rain, **R**eliz**A**tion, and **P**erformance), a framework to build adaptive solutions to the task of developing Network Intrusion Detection Systems (NIDS). We state a clear separation in capturing high-quality malicious and bonafide traffics followed by a Machine Learning (ML) classification task. At this point, we advance and show the concerns involved in the realization of the ML models, analyzing the feasibility of implementation and performance evaluation. Therefore, the contributions of this work are:

- The AB-TRAP framework specification, which allows an end-to-end approach to building adaptive NIDS solutions.
- The performance analysis of different machine learning algorithms, aiming at the model realization in the target machine.
- Two study cases take into account local (LAN) and global (Internet) environments with kernel-space and userspace implementation.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3101188, IEEE Access

This paper's remainder is structured as follows. We describe the most relevant related solutions in Section II. In Section III, we describe the AB-TRAP framework. We describe the validation of AB-TRAP and the results of two case studies in Section IV. Finally, we summarize our work and provide direction for future works in Section V.

II. BACKGROUND AND RELATED WORK

A. INTRUSION DETECTION SYSTEMS (IDS)

Intrusion Detection Systems (IDS) are responsible for identifying malicious activities. It may refer to a broad class of systems whose input is a traffic source and decide to classify whether an instance is malicious. There are host-based and network IDS, the former receive local information, whereas the latter has access to global data. Identification of the malicious activities may consider network packets isolated (stateless) or take a set of packets that form a flow [7]. Therefore, the application requirement influences how protection on the target system is, which includes tackling the IDS locality in decisions.

As argued in [7] and [8], approaches to the identification problem include signature[1]- and anomaly-based. Knowing the attacks' pattern helps to trace profiles in packets and communications, thus forming a signature of the network's malicious packets. On the other hand, for specific applications and well-known services, the distinction between normal and abnormal patterns leads to the decision when anomalies appear unexpected. Hence, establishing how hostile the operational environment favors exclusively one approach or other, or a hybrid one.

The stochastic nature of network traffic patterns and the packet features shift the modeling to statistical, Machine Learning (ML), and other techniques [7]. When defining a task to incorporate in IDS solutions, these classes of algorithms are responsible for classification. Thereby increasing the information gain based on datasets, they represent the core component to the system's decision-making process.

B. MACHINE LEARNING (ML) ON NIDS DOMAIN

The usage of old datasets for network security research is notorious [9], being challenging to tackle this issue, as described in [6]. The use of old datasets on NIDS research stills a concern as confirmed by [10]–[13], which reports the adoption of DARPA 1998, KDD-CUP '99, and NSL-KDD as the most-used ones, yet these datasets are around two decades old by now. An important factor in favor of reusing old datasets is applying Machine Learning (ML) algorithms to provide new approaches. From this perspective, well-established and available datasets allow benchmarking the performance of different solutions. Meantime, old datasets get aged and become obsolete for NIDS development and research as we observe the emergence of new types of attacks and network communication architectures. Privacy issues also prevent the availability of new public datasets [12]. Therefore, creating novel mechanisms to incorporate new technologies and cleaning the sensitive data from real-world traces balances keeping the existing dataset and constant updates.

A survey on NIDS datasets is provided by [14] that highlights the challenges of (*i*) outdated datasets in use by NIDS research, (*ii*) the characteristic of being unable to cope with the evolving pace of attacks, (*iii*) not using real-life traffic, and (*iv*) the limited number of specific purpose datasets (e.g., SCADA systems). Hence, a framework's adaptability is crucial to incorporate new features and updates to novel solutions [15]. As pointed by [12], the current datasets are not complete, and their utility is subject to their source. Further, they describe five categories of properties: nature of the data, volume, recording method, evaluation, and general information. The survey evinces the importance of testing solutions against multiple datasets to widen the case coverage, as labels and features help identify each set's screening profile. Thus, datasets play a crucial role in the proposal of NIDS solutions. Moreover, enable reproducibility and criteria for evaluation.

Machine learning (ML) algorithms are sensitive to the application-layer and protocols-specific traffic patterns. Under the Industrial Internet of Things (IIoT) context, [17] carried a study focused on SCADA systems. They were able to identify backdooring and SQL injection attacks using the clustering strategy. However, the results are too specific to that systems. Therefore, the combination of domain-specific dataset and ML provides insights into the solutions' development process and requires care to avoid overfitting.

Despite observing the dataset and correctly setup ML, patterns and attacks have a lifespan; that is, they are time-varying. As shown by [18], a study identified a six-week of good performance in the trained models, and the accuracy standard lasts from two to eight weeks. That study issues a vital discussion on the NIDS solutions' realization, indicating it is a process, not a software artifact deployed on the facilities. Nevertheless, their solution is resource-intensive, requiring CPU power, outbound network channels for analysis and storage.

Regarding the concern of not up-to-date datasets, [19] proposed a method to generate a dataset using a simulated environment and to use an anomaly detection approach to evaluate the detection performance for known, similar, and new attacks. However, the authors reported the inability to detect new attacks on the evaluated scenarios.

From the IoT domain, [16] presents a survey of state of art in IoT intrusion detection by the time. In addition to the more surveys previously discussed here, the authors report the unsuccessful task of collecting the artifacts for reproducibility and cross-validation of the papers and highly recommend open access. The [20] uses a dedicated dataset called Bot-IoT [21] in a hybrid approach, using both anomaly and signature-based NIDS. This hybrid approach first evaluates if a network flow is anomalous. If yes, then this object is forwarded to the attack classifier. Despite considering the applicability of the proposed framework for IoT devices

[1] Signature-based is commonly referred as misuse-based

TABLE 1: Attributes presented on the evaluated IDS surveys as research challenges (✓: Yes, Empty: No)

Reference	Focus	Old Datasets Criticism	Need of Labeled Data	Traffic Diversity	Reproducibility	Operational Overhead	Real-world performance	Interpretability
[7]	Techniques and datasets	✓	✓	✓				
[8]	Data Mining and ML	✓				✓		
[9]	ML					✓		
[10]	ML and DL	✓				✓		
[11]	ML and DL	✓				✓	✓	✓
[12]	Datasets		✓		✓			
[13]	ML and DL	✓				✓	✓	
[14]	Datasets	✓		✓	✓		✓	
[15]	Methods					✓		
[16]	IoT			✓	✓			
AB-TRAP	ML and Datasets	✓	✓	✓	✓	✓	✓	✓

and being a promising hybrid approach, the authors do not present the realization and performance evaluation of the solution to confirm their claims. It is also dependent on others datasets to continually evolve its solution. [22] generates its dataset using two physical testbeds, one using IP cameras and the other with multiple IoT devices in a wireless network. The authors proposed an anomaly-based NIDS to detect the attacks performed on these testbeds and evaluate the autoencoder's performance in a Raspberry Pi with success. However, the developed dataset is very tied to the built testbeds and difficult to reproduce.

The balance of high-quality datasets, properly trained models, case coverage, and resource consumption is challenging. As described by [23], datasets are the entry-point for such an equation and present multiple trade-offs. The leading utility in this context is the fostering of prediction. Hence, that servers as guidance to derive lightweight solutions suitable for devoided computing power devices. However, timed and a span of other types of attacks impact the performance of ML-based solutions [24]. Salting bonafide traces with attack variations expose weaknesses in models and support novel approaches to overcome them. Therefore, the combination of bonafide and malicious attacks allows flexibility to explore new patterns and adapt to the ML training phase.

C. IMBALANCED DATASETS

Imbalance in datasets for Network Intrusion Detection Systems is notorious for decreasing the effectiveness of Machine Learning (ML) based solutions. Different algorithms perform better or poorly when deployed depending on how discrepant the lack of adequate proportion of bonafide (legitimate or benign) and malicious. [27] present criteria for selecting proper datasets; however, the results are too specific for the application's context and hard to generalize. When shifting to more specific domains (e.g., Industrial), the imbalance problem becomes more evident. As a reference, the survey about IDS datasets conducted by [12] reports only 1 balanced dataset out of 34. The results from [25] show how sensitive models are when facing changes in such an imbalance; therefore, this is a vital characteristic to consider when evaluating or producing datasets. This problem is recurrent, and we see an increasing interest in this topic [29]. Many of the issues were addressed and still have many challenges to tackle. It is a vast area of study and not entirely concentrated in NIDS. Therefore, being out of our work's scope, the appreciation of how imbalance influences the models.

D. FILLING THE GAP TO REAL-WORLD

The previously mentioned characteristic of timely-evolving security plays a fundamental role in developing solutions to mitigate new attacks and identify new network traffic patterns.

The [13] states the low performance of NIDS in a real-world environment as a research challenge. Some of the probable causes defined by [13] are the use of old datasets and not evaluating the proposed solutions considering a realistic environment. Also, the resources consumed by complex models for deep learning impact the adoption of these solutions. In addition, the authors report as research challenges: the lack of a systematic approach for dataset generation capable of being updated frequently and the focus on lightweight IDS solutions aiming to secure IoT environments with sensor nodes with limited resources.

Complementing this discussion, the anomaly- versus signature-based approaches illustrate the contrast between the real-world NIDS solutions widely deployed, such as Snort[2], and Suricata[3] that are signature-based. The current researches on anomaly-based NIDS with questionable performance for new attacks as discussed by [6], evidenced by [19] for new attacks class. It is reported by [30] that some companies avoid anomaly-based systems. The survey conducted by [9] evaluated the references according to its approach: anomaly-, misuse- (signature) or hybrid-based with a majority of signature-based IDS (26 out of 39).

The trade-off of the signature-based approach is the good performance to detect known attacks on the cost of difficult to detect new attacks (zero-days). On the other hand, the anomaly approach can detect new attacks but suffers from high false positives rates. We use known attacks as part of our ML approach that could learn the patterns of the known attacks beyond the traditional rule-based learning, leading to satisfactory performance for similar attacks as reported by [19].

Another concern of NIDS solutions is using the method based on packet payload data (e.g., deep packet inspection). This method becomes unsuccessful due to the increasing presence of encrypted network traffic. Regarding attack tech-

[2] Snort: snort.org
[3] Suricata: suricata.io

TABLE 2: Comparison with related work that implements a NIDS (✓: Yes, Empty: No)

Reference	Dataset Source	Implementation Feasibility	Operational Overhead	Adaptability	Attack Scope Narrow	Remark(s)
[17], [25]	Testbed				Backdoor, Command and SQL Injection	The authors details the testbed, however, the concern about implementation and operational overhead is not presented.
[18]	MAWILab [26]	✓	✓	✓	Traces labeled as anomalous or suspicious	The authors presents the implementation and operation overhead in the big data perspective which requires high computational power that contrasts with devices deployment envisioned by this work.
[19]	Testbed				DoS, probing, vulnerability scan	Besides the dataset is available, this work do not make available the means to generate new datasets from the proposed testbed (lack of adaptability), neither discuss implementation feasibility and operational overhead of the models.
[20]	Bot-IoT [21]				DDoS, DoS, Scan, and Theft	Bot-IoT dataset is available, but it is not an evolving initiative (lack of adaptability). Operational Overhead is envisioned but not measured and no implementation is presented.
[22]	Testbed	✓	✓		MitM, DoS, Malware, Reconnaissance	Kitsune generates the dataset and validates its model based on surveillance cameras and IoT testbeds. Regarding adaptability, these testbeds do not allow the generation of new datasets, neither are easily reproducible. Uses a Raspberry Pi for implementation feasibility and operational overhead evaluation.
[27]	CSE-CIC-IDS2018 [28]				Bot, Bruteforce, DoS, Infiltration, SQL Injection	The authors do not discuss the operational overhead for IDS deployment, neither the model realization (implementation feasibility). Also the dataset used is not an evolving initiative (lack of adaptability).
AB-TRAP	Testbed and MAWILab [26]	✓	✓	✓	✓ (TCP port scanning)	AB-TRAP provides instructions to generate up-to-date datasets (adaptability), presents implementation feasibility (LKM, user-space application) and operational overhead evaluation with a Raspberry Pi.

niques, the use of VPN and SSH tunneling to evade firewalls and NIDS [31] evidences the flaw in current solutions. To address this concern, the use of ML raises as a feasible technique to learn the patterns without requiring inspecting packets' payload data. This technique had been proven useful for mobile devices as reported by [32], [33]. Also, we have the concept drift of ML models [34] with NIDS making unreliable predictions over time [18]. It is an inherent learning challenge to determine anomalous patterns in a specific dataset and generalize them to other devices and networks. A hybrid approach as presented by [20] and addressing concept-drift with periodical model updates are good paths to fill the gap to the real world too.

The adaptive nature of the attacks and traffic pattern requires various areas to support novel approaches in the NIDS solution context. For example, the assumption of a completely safe perimeter is not reasonable, and shifting to a zero-trust architecture [35] can increase network assets' overall protection. Simultaneously, the adoption of resource-intensive solutions (such as dedicated IDS with outbound infrastructure) and lightweight modules to offer invisibility or mitigation to malicious attacks offer protection to the network. Thus, the creation of less resource-intensive modules enables a better protection solution to hybrid NIDS (classic and distributed).

E. SUMMARY

As we see, modern solutions for Network Intrusion Detection Systems (NIDS) are complex and involves the combination of multiple areas to provide adaptive features. Machine learning-based detection emerges as a trend in current implementations as providing a feasible framework for classification. The literature review highlight the following aspects as research challenges: up-to-date datasets, labeled data, traffic diversity, reproducibility, operational overhead, real-world performance, and interpretability with the evaluated surveys summarized in Table 1 and we keep all these attributes on AB-TRAP framework. From the IDS implementation perspective, the aspects of dataset quality, generality in models, adaptability, implementation feasibility, and operational overhead are of major importance. Mechanisms as over-the-air updates (OTA) and low-resource machine learning models (such as TinyML [36]) are usually not seen by the literature. However, they play an essential role in the models' realization, becoming an impediment when models are incompatible with the target hosts' requirements.

Therefore, our work differs from others by mapping the first attempt to approach new attacks until the mitigation module deployment. We describe it as an end-to-end framework to implement software artifacts in the iterative process of protecting the network perimeter. Our solution offers the means to inspect and improve data quality and case coverage by specifying a step for systematically reproducing or collecting malicious activities and mixing them with bonafide traffic. The separation of datasets and models is crucial and present in our solution, enabling different models' specifications, each with peculiarities and different learning metrics. Performance in the modeling phase is not the guarantee of realization in the target system. Depending on the model characteristics, it will require computational resources that are infeasible for the target. Thus, our solution requires the model realization followed by the computing system performance evaluation. All the steps in our framework form a pipeline. The first input is the traffic, and the last output the protection module; moreover, it is suitable to be part of constant evolving monitoring of network threats. Thereby, our framework fills a gap in the adaptive process of developing NIDS solutions. A comparison between the reviewed works focused on implementation and their characteristics in comparison with AB-TRAP are present in Table 2. We also consider the attack scope narrow as an attribute of comparison based on [6] recommendations for using ML in the IDS domain. To the best of our knowledge, the AB-TRAP is the only framework addressing adaptability with a systematic approach for up-to-date dataset generation capable of evolving with attacks changes, also addressing the model realization and its computational performance evaluation to support the real-world application.

III. AB-TRAP FRAMEWORK

This section describes our solution to enable an iterative and end-to-end approach to derive protection modules. We present AB-TRAP (**A**ttack, **B**onafide, **T**rain, **R**ealiz**A**tion, and **P**erformance), a framework comprising steps to build Attack and Bonafide datasets, train machine learning models, realize (implement) the solution in a target machine, and evaluate the protection module's performance. Figure 1 depicts the AB-TRAP's development chain. The design decisions during its use are also presented, such as a dataset change required over the training of machine learning models or changes required after the non-compliance with the performance evaluation. These new requirements on the performance evaluation can lead to changes in the training phase (e.g., use other algorithms) or the approach used by the realization step (e.g., kernel-space vs. userspace deployment).

A. ATTACK DATASET

Our framework starts with a method to generate a dataset containing the attacks. The objective is to overcome the available datasets' challenges: not containing the most recent attacks, not reproducible, and not properly labeled. Our analysis from a recent datasets' survey [12], restricted to the port scanning problem (presented in our study cases), started from 34 documented datasets, after considering: only those with packets granularity, and labeled datasets, resulted in just five datasets, but none of them presented the most recent tools, and techniques for port-scanning, such as Internet-wide scanners like Zmap [37], and masscan [38] or not enough information was public to determine all attacks present on the dataset. This analysis illustrates the challenges pointed out as the purpose of this framework.

In this paper, we detail the generation of attack datasets based on network packet granularity. However, it is possible to take advantage of the proposed methods using virtual machines (VMs) and containers to create new attack datasets based on network flow, and also for host-based IDS (HIDS), using logs and operating system measurements (e.g., processes; I/O usage).

B. BONAFIDE DATASET

The bonafide dataset contains examples of legit behavior. For the supervised learning approach, it is mandatory to have this dataset. It is also essential for the evaluation of both supervised and unsupervised learning algorithms. This dataset can be retrieved from open-data traffic providers as MAWILab [26], from public datasets as [19], or generate the bonafide dataset considering the deployment environment for the trained model. The third option is the most difficult to perform and the most successful because the bonafide traffic behavior can change from one environment to the other and from time to time. In all cases, we see the critical and challenging task of correctly labeling also applies.

The proposed AB-TRAP dataset generation workflow is known as salting [24], combining legit and attack data. Obtained from real traffic traces and synthetic datasets, keeping in mind the objective to address the current issue with NIDS research using outdated and unlabeled datasets [6] [14] [23].

C. TRAINING MODELS

This AB-TRAP step is responsible for creating machine learning (ML) models using the previous steps' datasets. For the ML application on network intrusion detection systems (NIDS), the essential points to highlight are the assessment metrics and the envisioned realization of these ML models. Regarding the assessment metrics, the most used from the ML domain is the accuracy. Though, once the NIDS is inherently an imbalanced learning problem, we use the F1-score, precision, and recall as the assessment metrics for selecting the trained models with precision representing the exactness, and recall its completeness, then F1-score is the weighted harmonic mean between precision and recall. These metrics use the values retrieved from the traditional confusion matrix (Table 3), such as True-Positive (TP), True-Negative (TN), False-Positive (FP), and False-Negative (FN):

$$\text{Precision} = \frac{TP}{TP+FP}\,,\ \text{Recall or TPR} = \frac{TP}{TP+FN}\,,$$

$$\text{F1-score} = 2 \times \frac{\text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}}\,.$$

TABLE 3: Confusion Matrix (+: Positive Class, – : Negative Class)

		Predicted Class	
		+	–
Actual Class	+	True Positive (TP)	False Negative (FN)
	–	False Positive (FP)	True Negative (TN)

F1-score has a high correlation with receiver operating characteristic (ROC) and the area under the ROC curve (AUC). However, we also use ROC/AUC to support the trade-off during the development of the models. This ROC/AUC use the recall, also known as sensitivity or true positive rate (TPR), and the false-positive rate (FPR), also known as false alarm:

$$FPR = \frac{FP}{FP+TN}\,.$$

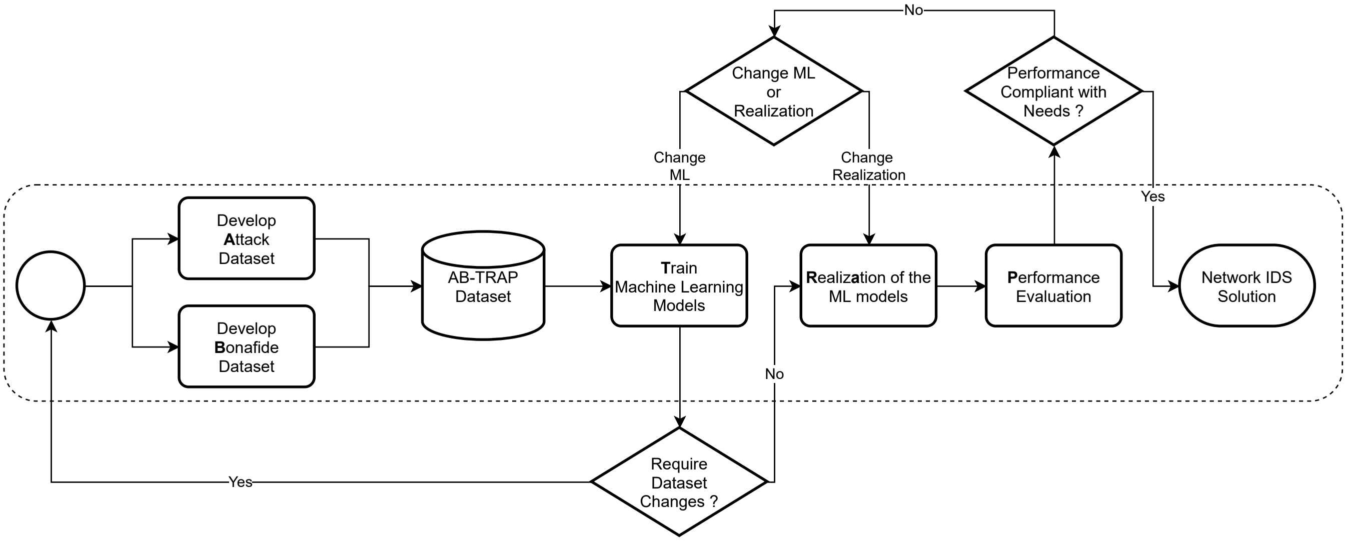


FIGURE 1: AB-TRAP Framework

One lesson learned over the framework development is related to the challenges for the realization of ML models. In the NIDS domain, there is an opportunity to perform the intrusion detection in the network core, with solutions operating at a higher level (e.g., userspace). On the other hand, the classification algorithms could operate directly in the end-node at a lower level, implementing it in conjunction with network stack software or directly in the network interface cards (NIC) through SmartNICs/FPGAs [39]. In both cases, the implementation of trained models is a challenge. Thus, we consider ML algorithms that are interpretable [40] to support the realization step of AB-TRAP when considering kernel-space deployment.

D. REALIZATION

The realization step starts from the trained model selected in the previous step and requires implementing a computational system's trained model. Further, we must consider a highly coupled solution between the implementation and deployment environment. Thus, we present here some possibilities. For the software-defined networks (SDN) / OpenFlow environments, commonly associated with higher computational resources, the deployment is associated with a holistic view of the network environment. For instance, [41] proposes an IDS deployment in multiple stages, from the cloud down to IoT gateways. The cloud-based IDS have elastic resources, so no rigorous attention is required for realization (e.g., it is acceptable to use deep learning methods). In the middle of the traffic path on the fog environment, the deployment in the SDN controllers, which generally also provide sufficient computational resources, and finally IDS tasks in the SDN-based IoT gateways also have enough computational resources for complex processing. The IoT devices in the perception layer are out of scope as a deployment target.

Considering the IoT device in the perception layer that is usually resource-constrained, we must take this inherent characteristic as requirement for the realization step. First of all, it is possible to envision the deployment of IDS rules in these devices from a zero-trust architecture perspective. Thus, along with the lack of resources, they are unable to evaluate their network perimeter. For those microcontroller-based devices, the realization of the ML models must be along with the network stacks used by them in a lightweight way once power, processing, and memory are constrained.

For those devices using embedded Linux, the most promising approach takes advantage of kernel-level processing. In these cases, it would be possible to use Linux Kernel Modules (LKM) along with Netfilter for packet filtering. This approach is possible since Linux kernel 2.4 released in 2000. The approach with LKM and Netfilter processes each packet received in the network interface during its traversal path up to the application layer. Another realization approach for Linux systems is through eBPF/XDP (extended Berkeley Packet Filter / eXpress Data Path), eBPF released in the kernel 3.18 using a virtual machine available in Linux Kernel that allows the execution of byte-code safely direct in the kernel space. The XDP available since kernel 4.8 enables the eBPF code to run directly in the network driver or on the SmartNICs, making the process at the earliest point of the network traversal path. Our preliminary analysis in a single board computer with an embedded Linux presents an increase of network throughput when using eBPF/XDP instead of LKM/Netfilter [42].

E. PERFORMANCE EVALUATION

Our final step in the AB-TRAP aims to evaluate the developed NIDS solution's performance in the deployment environment. This performance evaluation is part of the framework because the trained ML model may have a resource demand or create a bottleneck that makes it a deterrent for real-life use. There are two major groups for performance metrics, those based on the network performance and the metrics based on the device performance. We point to the throughput measurement for both total bytes and the number of packets per second for network metrics. For devices, the power consumption, CPU usage, and RAM consumption during NIDS model operation compared with the baseline could be another NIDS solution or vanilla configuration.

IV. STUDY CASE: PORT SCANNING DETECTION

In this section, we present the AB-TRAP framework addressing port scanning detection. The port scanning detection is a strategic decision as it enables the attack interruption in its early stages, also known as reconnaissance phase by security frameworks. Studies performed on 7.41 Petabytes of network traffic, collected in a network backbone between 2004 and 2011, raised that about 2.1% of packets are scan [43].

We exercise our framework in two contexts: firstly in a local area network (LAN) to address networks not open to the Internet. This context is typically associated with lateral movement during attacks and with malware spread inside enterprise networks; secondly, to address the port scanning issue in the Internet environment, that is normally associated with massive scanning to identify accessible assets over the internet, or in conjunction with the release of new remote vulnerabilities to identify non-patched systems. For both study cases, we consider a stateless approach for port scanning detection.

A. LOCAL AREA NETWORK (LAN)

1) Attack Dataset

The first step is always obtaining a dataset of the *positive examples* for our learning task. In this case, it is a dataset containing network packets of port scanning in a LAN environment. This paper uses a reproductive and automated approach that takes advantage of virtualization methods and the LAN environment. We use Vagrant[4] to create multiples virtual machines (VMs), as detailed in Figure 2.

This architecture consists of an attacker subnet (`172.16.0.1/32`) with an *attacker VM* running on Kali Linux[5] with the most-updated port scanning tools and using a provisioning shell script that executes all the possible combinations of attacks to the *target VM* in the victim's subnet (`10.10.10.1/32`). The provisioning script considers a specific IP address for each of the attacks performed, which serves as a label to the data. Furthermore, we configure two VMs with static routes to interconnect the subnets, and *router0* is responsible for capturing all the traffic between both subnets. Table 4 summarizes the scope of port scanning tools and techniques we use. The techniques are those that use partial or full three-way handshake from TCP protocol, TCP SYN and TCP Connect, respectively, or flag-based techniques that use specific behaviors of the TCP implementation by the operating systems for malformed packets/flags (in comparison with RFC 793) to infer the state of the services in the target machines.

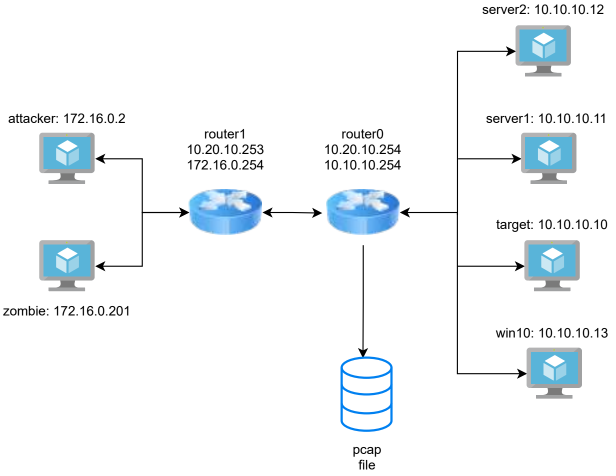


FIGURE 2: LAN Environment

TABLE 4: Port scanning tools and techniques for TCP Scan

	3-WAY HANDSHAKE		FLAG BASED					
	SYN	Connect	FIN	XMAS	NULL	ACK	Window	Maimon
Nmap	✓	✓	✓	✓	✓	✓	✓	✓
Unicornscan	✓	✓	✓	✓	✓	✓		
Hping3	✓	✓	✓	✓	✓	✓		
Zmap	✓							
Masscan	✓							

2) Bonafide Dataset

For the bonafide dataset, we make use of the MAWILab dataset [26]. It is 15-minutes daily traffic from a trans-pacific backbone between the USA and Japan. The process to obtain the bonafide traffic is based on the premise that for each MAWILab dataset, the labels of *anomalous* and *suspicious* traffic are also available. From that, we perform a filtering step to remove all packets based on these labels/rules. The rules consider source/destination IP, source/destination ports and provide a specific taxonomy for each non-bonafide traffic. In conjunction with this information, reports a heuristic and distance measurement from [26] classifiers to determine the traffic as anomalous or suspicious based on a combination of four various detectors. We obtain the bonafide dataset that is merged with the attack dataset from the previous step to obtain our AB-TRAP dataset from this process. This dataset generation flow is presented in Figure 3.

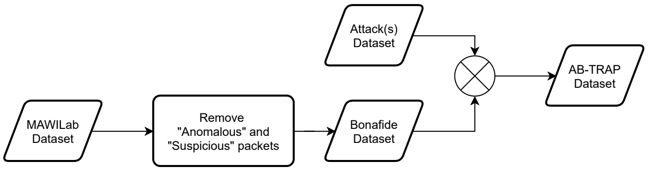
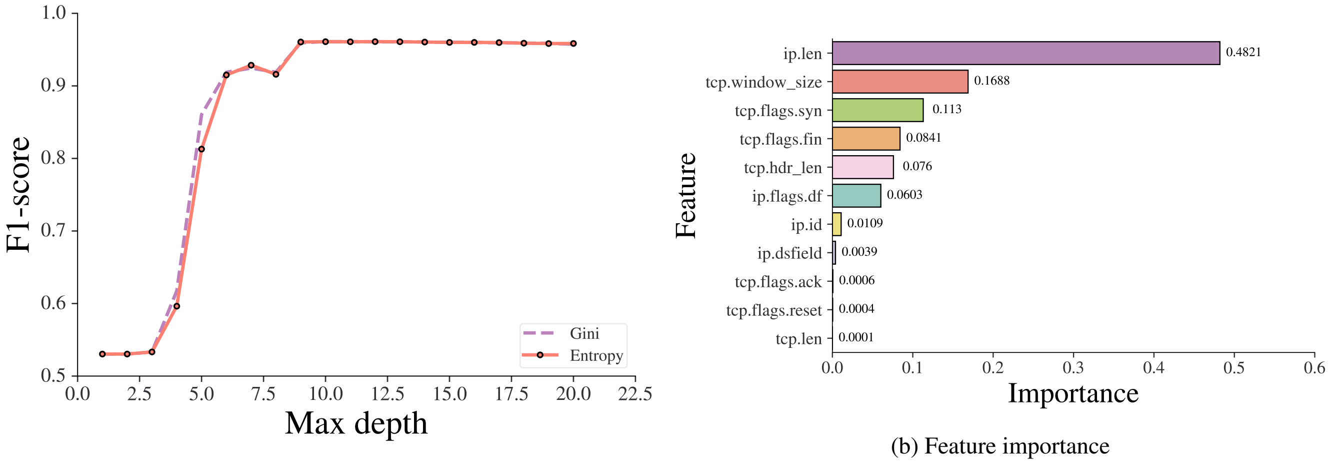


FIGURE 3: Composing AB-TRAP dataset with bonafide data

[4] Vagrant by HashiCorp: vagrantup.com

[5] Kali Linux: kali.org



(b) Feature importance

(a) Performance of grid-search

FIGURE 4: Decision tree analysis

To obtain the bonafide dataset for this LAN study case, we used the MAWILab dataset of November 11th, 2019[6]. It is a 10 GB `pcap` file divided into multiple files to be managed by a regular desktop PC. After that, we remove all anomalous and suspicious packets according to its rules, then a randomized sampling of 75,000 packets from each split, resulting in a total of 2,141,635 packets and 184.9 Mb.

3) Training Models

From the AB-TRAP generated dataset (denoted as S), the first step is to address this supervised learning task as a binary classification problem, in other words, all attacks labeled by its tools/technique (Table 4) are re-labeled as "attack" or the positive class ($y_i = 1$), and the remaining packets as "bonafide" or the negative class ($y_i = 0$). The training dataset $S = (x_i, y_i), i = 1,...,m$, where $x_i \in X$ is an object in the n-dimensional feature space $X = \{f_1, f_2, ... f_n\}$, and $y_i \in Y = \{0,1\}$. The feature space is composed by all TCP/IP header features , and in this LAN study-case $m = 113{,}759$, with the subsets $S_{attack} = 22{,}373$ (18%), and $S_{bonafide} = 91{,}386$ (82%). From the original 2,141,635 packets, the bonafide subset of 91,386 packets is composed only by those from TCP/IP communication.

We perform a pre-processing of S to remove all non-applicable, redundant, and non-variant features in the sequence. As S cames from the `pcap`, we remove all layer 2 (Ethernet) features and keep only the TCP/IP features. Then we remove the IP header field version and protocol once they are constant, and IP source and destination fields are also removed not to associate the learning with the generated dataset, but to generalize the learning based on the other features. We remove the following IP fields due to non-variance: header length, type of service, fragment offset, more fragments, and reserved bit flags. We also removed the TCP TTL field once initial tests with this feature make the model learn the attack testbed architecture (Figure 2). Finally, the pre-processing results in a feature space of $n = 15$.

We use the Python language and the scikit-learn over this step. Then, we proceed with the evaluation of the machine learning models based on the F1-score criteria. In this study case, those algorithms are interpretable, and this assumption enables the next step of the framework that is the realization of the trained models as LKM/Netfilter module.

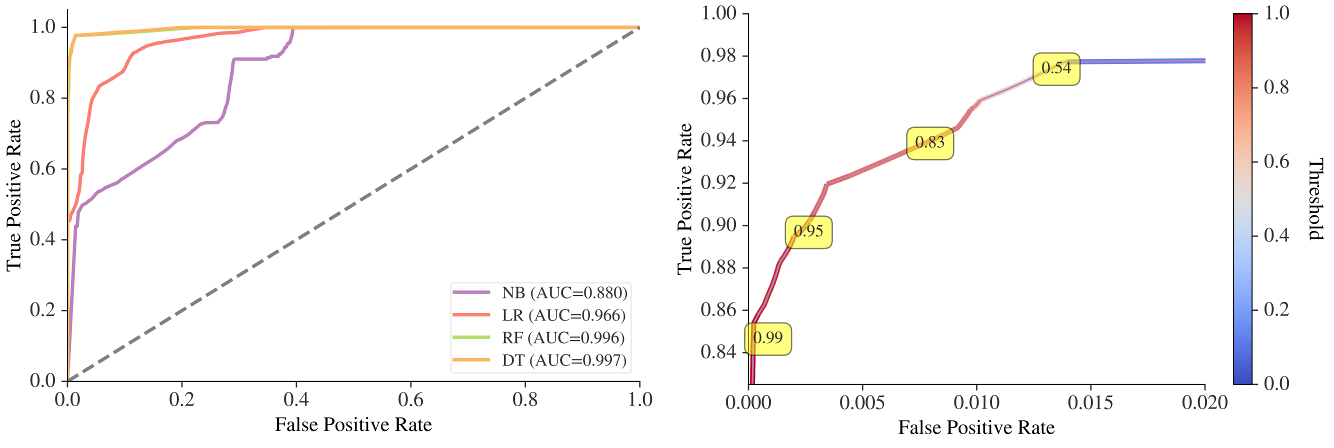
The algorithms are Naïve Bayes, Logistic Regression, Random Forest, and Decision Tree. Those with multiple parameters (random forest and decision tree) were trained using a grid search to determine the combination with the best F1-score performance. Random forest grid-search was performed with the number of estimators of [10, 50, 100, 200], the criteria of Gini, and entropy; max depth of [5, 10]; and using a class weight of None, balanced, and balanced subsample, for the decision tree was used the criteria of Gini, and entropy; max depth up to 21, and class weight of none, and balanced. The difference between random forest, and decision tree max depth is due to the number of estimators for the random forest leading us to reducing its depth.

All algorithms were evaluated in a stratified k-fold (k=10) setup. During this evaluation, the features were standardized by removing the mean and scaling to unit variance. The summary of the algorithm's performance is presented in Table 5.

TABLE 5: F1-Score for algorithms evaluated

Machine Learning Algorithm	Average F1-Score
Decision Tree (DT)	0.96
Random Forest (RF)	0.96
Logistic Regression (LR)	0.70
Naïve Bayes (NB)	0.55

[6] fukuda-lab.org/mawilab/v1.1/2019/11/11/20191111.html



(a) ROC curve and AUC for each model on LAN case

(b) Threshold analysis for Decision Tree (zoomed axis)

FIGURE 5: ROC/AUC analysis

We moved forward with the decision tree algorithm based on its best F1-score performance compared to the other algorithms (slightly better than RF). From the grid-search, we detailed the parameters' combination to determine that adopted in the realization step. According to Figure 4a, we confirmed that the F1-score increase at a max value of 0.96. In this case, we selected the shallow version of the decision tree with the F1-score of 0.96 to mitigate over-fitting. This decision results in a max depth of 11, with the Gini criteria and a balanced class weight.

It is important to note that the decision tree algorithms perform a feature selection during their training process, which is considered an embedded feature selection. Due to this characteristic, we also retrieve the feature importance from the trained model, allowing us to determine the most critical features for the classification task in this study case. Figure 4b presents the feature importance for those features with importance higher than zero.

We also compare the algorithms using a receiver operating characteristic (ROC) analysis. First, each algorithm estimates the positive class probability of all samples in the dataset. The estimates are calculated using a similar setup as before, i.e., 10-fold stratified cross-validation, however, the grid-search on the best hyperparameters optimizes the area under the ROC curve (AUC). Figure 5a displays the ROC curve and the respective AUC for each algorithm, with an average AUC of 0.95. The algorithms DT and RF shows the best performance overall.

The ROC analysis enables us to understand the expected true positive rate (TPR) and the expected false positive rate (FPR) when we change the threshold of the estimated positive-class probability to label a sample as positive. For instance, decreasing the default threshold 0.5 to 0.1 – that is, classifying as positive all samples that have estimated positive class probability greater than 0.1 – of the DT algorithm increases the expected TPR from 0.977 to 0.99 at the cost of also increasing the expected FPR from 0.014 to 0.124.

Moreover, in a real-world scenario, we can adapt the threshold to achieve a specific goal. The ROC analysis gives us information to select such a threshold and inform us about the drawbacks. For instance, if one needs to guarantee an expected FPR lower than 0.5% using DT, the appropriate threshold is 0.829 resulting in an expected TPR of 92.3% according to Figure 5b.

Also, in our experiments, NB cannot achieve such a low FPR while keeping the expected TPR greater than 0.

4) Realization

We considered our deployment in a single board computer (SBC). This decision aims to support the deployment of the port-scanning detection in devices with relatively low resources compared with regular PC on LAN and to be possible to extend to devices operating as IoT nodes and gateways. In this study case, the selected SBC is a Raspberry Pi 4 with the configuration detailed in Table 6.

TABLE 6: Raspberry PI 4 Model B specification

Component	Description
CPU	Cortex-A72 (ARMv8) 64-bit SoC, 1.5 GHz
RAM	4 GB SDRAM
Wireless	2.4 GHz and 5.0 GHz IEEE 802.11ac
NIC wired	Gigabit Ethernet
Linux Kernel	5.4.0-1032-raspi
Distribution	Ubuntu 20.04 LTS

The decision tree has the decisive advantage of being easily implemented as a combination of *if-then-else* statements. In this study case, we decided to implement the trained model as an LKM/Netfilter. The realization started from the trained *scikit-learn* model in conjunction with the *emlearn* [44] Python library to generate a generic C header file, then, a manual step was required to change the features references to the ones available on the Linux kernel header files. Listing 1 presents an excerpt of the implemented LKM.

Listing 1: Pseudo-code from decision tree

```
int check_packet(struct sk_buff *skb [...]){
  struct iphdr *ip = ip_hdr(skb);
  struct tcphdr *tcp = tcp_hdr(skb);
  ...
  if (tcp->window < htons(1)){
    if (ip->id < htons(60))
                return NF_DROP;
        else
                return NF_ACCEPT;
  }
  ...
}
```

5) Performance Evaluation

The last step of the AB-TRAP framework consists of the realized model's performance evaluation. The two characteristics under analysis are CPU usage and memory RAM utilization. We carried out the evaluation with two scenarios: with and without LKM loaded on the SBC. Then, we used the iPerf3 tool [45] to generate TCP/IP traffic over 20 minutes (1200 seconds) from host computer to the SBC with a sampling rate of 1Hz of the CPU and RAM usage. We use `sysstat` tool [46] as an observability tool to retrieve these metrics.

The performance evaluation of the LKM/Netfilter model for the decision tree with a max depth of 11, the Gini criteria, and balanced class weight is summarized in Table 7. Comparing the two configurations with and without LKM demonstrates an overhead minimal from RAM usage, and negligible about CPU usage in kernel mode. Tests performed with a Raspberry Pi 3, also confirmed this minimal overhead for a decision tree with max depth of 11.

TABLE 7: Comparison execution with and without the LKM for TCP Scan detection. The overhead is minimal.

	%memused		**%sys**	
n=1200	w/o LKM	with LKM	w/o LKM	with LKM
mean	6.20 ± 0.01	6.21 ± 0.01	9.47 ± 0.65	9.47 ± 0.61
max	6.24	6.32	11.05	11.02
min	6.15	6.18	1.00	1.26

Note: %memused - used RAM; %sys - CPU usage in kernel mode.

B. CLOUD ENVIRONMENT / INTERNET

This variant of the study case presents an environment change, using the Internet environment instead of the LAN. So, testing the framework in this new context for the stateless detection of port scanning.

1) Attack Dataset

In this new context, the infrastructure is created through cloud instances geographically distributed on the world and presented by Figure 6. One of the instances (us-east-1) is used as an attacker to perform the port scanning attacks (Table 4) against all the other instances.

The challenge for the attack dataset generation in the Internet environment is to coordinate the correct generation of the `pcap` files on each instance and correctly label each attack performed. Our approach for that is to create a service on each target instance communicating through UDP, this decision is not to conflict with the TCP/IP traffic, which is the scope of this study case. For each attack, the attacker instance sends a message to the target through UDP to start the `tcpdump`. Afterward, performing the port scanning activity, then a stop message is sent through UDP. This communication flow is illustrated in Figure 7. The target instances label its `pcap` files through a specific filename for each attack performed that uses a numeric reference and a timestamp.

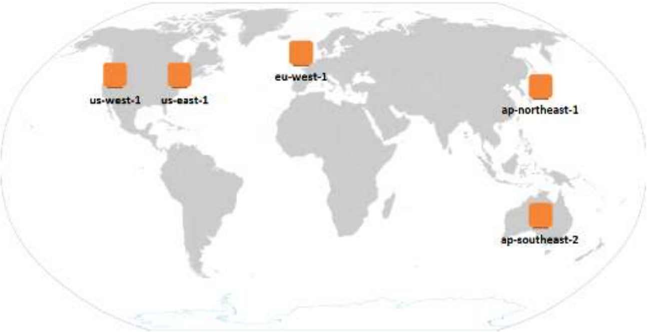


FIGURE 6: Cloud instances to be used as attacker/targets

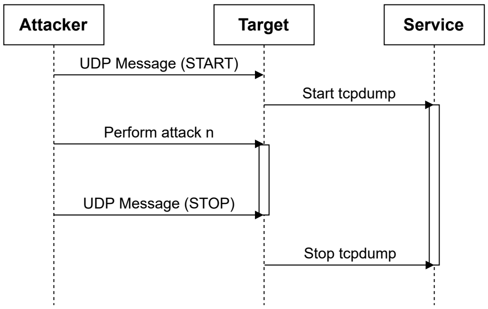


FIGURE 7: Lifeline of the attack dataset creation for each attack

For the attacker instance, we use a container from the Kali Linux docker image and a provisioning shell script that perform all port scanning attacks in conjunction with the UDP communication using the most recent versions of the tools (Nmap, Unicornscan, Hping3, Zmap, and Masscan) available in the Kali Linux repositories.

After attacks take place against all targets, we retrieve the `pcap` files from the instances and, using a Python script, merge all `pcap` in a single CSV file and properly labeled based on the filename. This CSV file is the attack dataset, composed of $455{,}503$ packets.

2) Bonafide Dataset

For this study case based on the Internet, we considered the same bonafide dataset used on the LAN case ($S_{bonafide} =$ 91,386). However, we merged the dataset with MAWILab

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3101188, IEEE Access

bonafide traffic from November 10, 2020[7], and from November 29, 2020[8]. After splitting, sampling, and filtering out just TCP/IP traffic, we got a total of 380,438 packets of bonafide traffic.

3) Training Models

The generated dataset (S) for the Internet case is composed of $n = 835{,}941$ packets with the following distribution, $S_{bonafide} = 46\%$ and $S_{attack} = 54\%$. In comparison with the LAN study case, this is a more balanced dataset. This class distribution will allow a better understanding of the performance achieved by each ML algorithm once the critical aspects of imbalanced datasets are not in place.

The criteria for pre-processing the dataset used here: we considered the two applicable layers' header fields (detailed in LAN study case) and removed IP source and destination addresses, including constant and non-variant features. Those header fields present in aggregate form (e.g., TCP flags) are considered only the discrete value (e.g., SYN). A difference from the previous study case is that we kept the TCP TTL field and source port as a feature. We use Python for the training step and the scikit-learn package for the F1-score criteria. However, we evaluate more ML algorithms in addition to those evaluated in the LAN study case, and they are Multi-Layer Perceptron (MLP), Support Vector Machine (SVM), k-Nearest Neighbours (kNN), and XGBoost.

We keep the validation step with the stratified K-fold ($k = 10$), and the scaling is the same as the one used on the LAN study case. All the grid search parameters and the f1-score for each k-fold are available in the repository. Table 8 summarizes the f1-score performance for all algorithms and shows the average from each k-fold, the standard deviation is negligible, and an overall average F1-score of 0.95.

TABLE 8: F1-Scores for algorithms evaluated

Machine Learning Algorithm	Average F1-Score
Multi-layer Perceptron (MLP)	1.00
k-Nearest Neighbours (kNN)	1.00
XGBoost (XGB)	1.00
Random Forest (RF)	1.00
Decision Tree (DT)	1.00
Support Vector Machine (SVM)	0.92
Logistic Regression (LR)	0.92
Naïve Bayes (NB)	0.78

The high f1-score such as 1.00 for MLP is a strong indication for over-fitting from the dataset's machine learning models. It is also important to highlight that the 0.92 f1-score for logistic regression is a good indication that a simple parametric model can be used to detect the internet use case. Despite the inherently correlation between f1-score and the ROC/AUC, the calculation is also performed through the estimation of positive class probability for each algorithm based on all samples in the dataset. This decision aims to support further discussion on ROC analysis regarding the threshold and TPR/FPR trade-off. Figure 8 displays the ROC curve and the respective AUC for each algorithm, with an average AUC of 0.98.

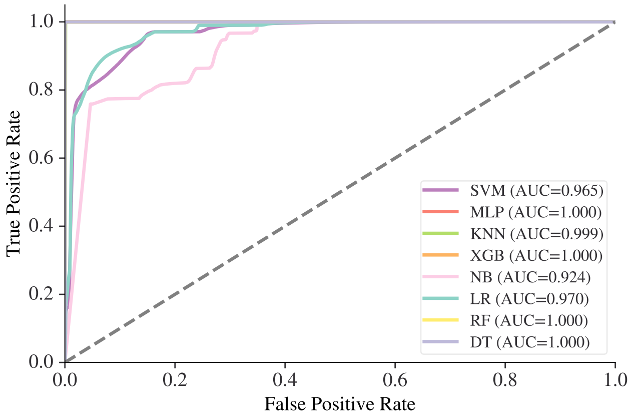


FIGURE 8: ROC/AUC analysis for Internet case

All these points can be further analyzed from a data science perspective. For this case, we move along considering the next steps of the framework, the realization and the performance evaluation, without digging deeper on the training models step.

4) Realization

In comparison with the previous use case, on this one, we are still considering the computational device as an SBC. But, instead of deploying the machine learning model in kernel-space using LKM, we use a different approach to deploy the ML models in the userspace. This approach enables using a multitude of machine learning models abstracting the complexities to implement them on kernel-space.

The NIDS relies on the functionality available on the `iptables` firewall using the Netfilter NFQUEUE module to enable the decision about the packet in userspace. In our case, all incoming TCP packet flows from kernel-space to userspace, and after that to the ML model, this packet traversal path is shown in Figure 9. The development of the NIDS in this use case requires transforming the machine learning models into the "final models," the concept of the final model is the model trained on all available datasets with the chosen parameters from grid-search on the training step. After that, each model becomes a byte stream. An application developed in Python receives these extracted models' deployment and executes them in the userspace with elevated privileges to execute with NFQUEUE.

5) Performance Evaluation

For the performance evaluation of trained models in this step, we used the SBC detailed in Table 6.

[7] fukuda-lab.org/mawilab/v1.1/2020/11/10/20201110.html
[8] fukuda-lab.org/mawilab/v1.1/2020/11/29/20201129.html

TABLE 9: Metrics for each trained model

Machine Learning Algorithm	Storage Usage	Inference Time (ms)	CPU Usage (%)	RAM Usage (%)
k-Nearest Neighbours (kNN)	108 MBytes	313.57 ± 1.59	3.07 ± 3.75	22.61 ± 5.91
Random Forest (RF)	223 KBytes	18.56 ± 0.03	4.01 ± 0.96	11.17 ± 0.55
Naïve Bayes (NB)	1.3 KBytes	4.19 ± 0.02	1.17 ± 1.27	10.82 ± 0.47
Multi-layer Perceptron (MLP)	16 KBytes	4.16 ± 0.01	1.48 ± 1.08	10.79 ± 0.50
Support Vector Machine (SVM)	890 Bytes	3.92 ± 0.01	1.50 ± 1.07	10.95 ± 0.51
Logistic Regression (LR)	939 Bytes	3.92 ± 0.01	1.14 ± 1.27	10.65 ± 0.49
Decision Tree (DT)	13 KBytes	3.97 ± 0.01	1.89 ± 1.58	11.90 ± 0.63
XGBoost (XGB)	279 KBytes	3.23 ± 0.01	1.72 ± 1.06	10.79 ± 0.53
Baseline (None)	–	–	0.55 ± 0.20	8.88 ± 0.01

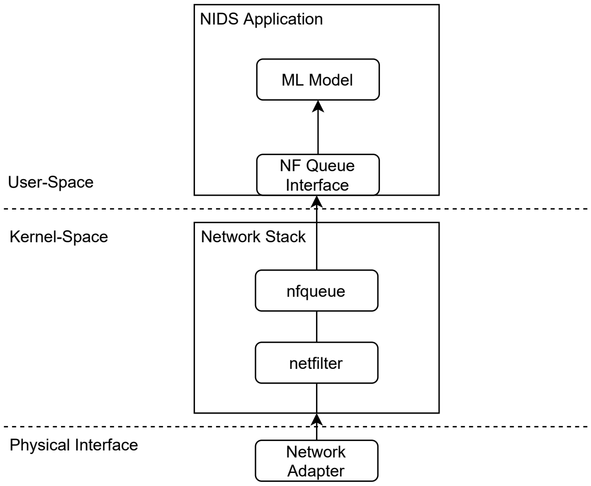


FIGURE 9: NIDS packet traversal path

Notwithstanding the SBC having a gigabit Ethernet interface, in this performance evaluation, the host computer interfacing with the SBC has a 100 `Mb/s` Ethernet interface, so the nominal interface speed between a host computer and target device is limited to 100 `Mb/s`. Similar to the LAN study case, we are using the iPerf3 tool [45] to generate TCP/IP traffic for 20 minutes and 1Hz sampling with each of the eight trained models loaded at once. We deployed a UDP daemon on the target device to manage this performance evaluation. Each UDP message was responsible for starting the iPerf3 server, setting the `iptables` rules with `nfqueue`, loading the NIDS application with the specific ML model under evaluation, and starting `sysstat` monitoring. A shell script was used from the host computer side to generate the eight models' messages under evaluation.

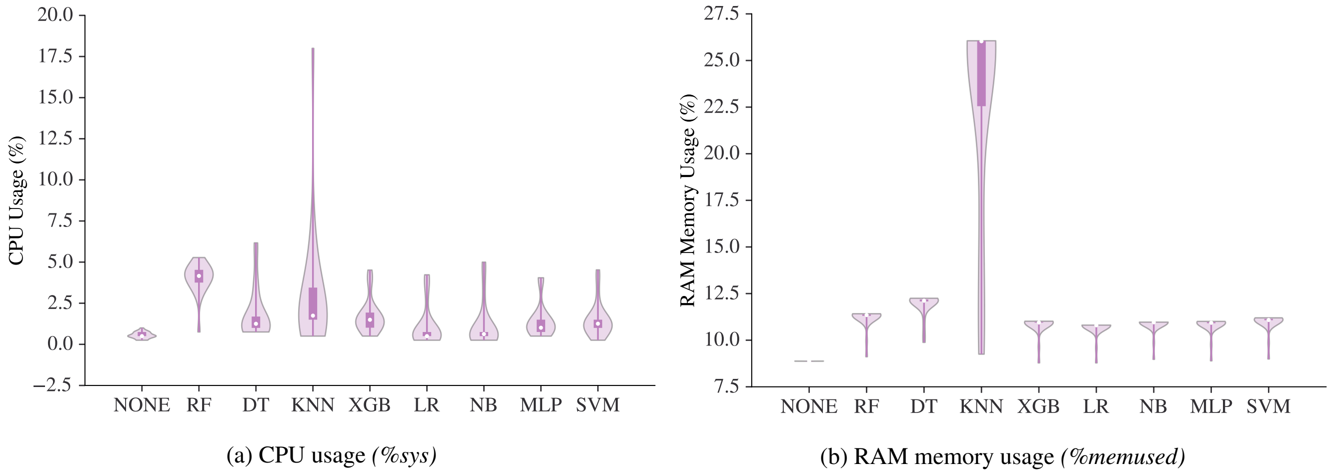
For each of the models during the 20 minutes test, the following performance metrics were measured: CPU usage and RAM usage. The performance of each model is presented in the Figure 10a and Figure 10b.

From the CPU usage perspective, KNN, RF, and XGB are the models that require the most processing power during the operation of the NIDS. Regarding RAM usage, KNN stands out as the most resource consumer model followed by the random forest; the other models do not present significant differences.

For this study case, we considered two additional metrics: storage usage and inference time. The first is a helpful metric to determine the trained model's deployment feasibility on an embedded device. In addition, the latter is an important metric to consider for the specific task that the ML application envisioned. The storage usage was determined from the extracted model size on the SBC file system. The inference time was obtained from a workflow consisting of creating a packet with random attributes (20 times) in conjunction with Python's *timeit* library for averaging 1000 inference executions over these random packets. Table 9 summarize the performance for all trained models in conjunction with a baseline (referred as "None" on Figures 10a and 10b) that represents the SBC without a ML-model loaded and using an iptables rule to simply block the packets from iPerf3 in the baseline scenario. Summarized as an average overhead of 1.4% CPU, and 3.6% RAM considering the difference between each model and the baseline.

From the performance results, it is possible to confirm the negative aspects of KNN for embedded applications and packet processing. It is a slow algorithm and does not create a compact model based on the data. Instead, it requires the complete dataset as part of the model. Such a characteristic makes the training step more relaxed from using computational resources. However, the new sample has its distance calculated with all available examples for the inference phase, making it resource-intensive when implementing. In comparison, RF model also presented an inference time that can become unfeasible for certain applications. The simpler models such as LR and NB presented minor impacting results concerning the use of computational resources. Notwithstanding, for the NB, we observed poor performance evidence compared with the other algorithms as detailed in Table 8.

We get to a multi-objective trade-off; we have the performance metrics based on the computational resources and the model performance from the data science perspective (F1-score, ROC/AUC). Also, this decision is the last decision step from this framework represented in the Figure 1 by *"Perfor-*



(a) CPU usage *(%sys)*

(b) RAM memory usage *(%memused)*

FIGURE 10: Performance evaluation for each ML Model

mance Compliant with Need?". If successful, it moves to the deployment of the NIDS solution; otherwise, getting back to reviewing the ML model or its realization approach.

V. CONCLUSION

In this paper we present AB-TRAP (**A**ttack, **B**onafide, **T**rain, **R**ealiz**A**tion, and **P**erformance), a framework comprising steps to build Attack and Bonafide datasets, train machine learning models, realize (implement) the solution in a target machine, and evaluate the protection module's performance. One of the main concerns in implementing effective Network Intrusion Detection Systems (NIDS) is the ability to adapt to new attacks and the evolution of the network traffic. AB-TRAP systemizes the whole chain of design and implementation of NIDS solutions and forms a pipeline to build protection modules for the constant-evolving malicious activities. Moreover, we highlight that AB-TRAP is part of an iterative process where the network traffic source (malicious or bonafide) is the input and the updatable modules (e.g., suitable for Over-the-air update or Linux Kernel Modules) are the output.

We test AB-TRAP in two environments: LAN and the Internet. In both cases, we achieve low-resource utilization protection modules, and Decision Tree provides the best performance for the training and realization phases. In the first case study, our results show an f1-score of 0.96, and the overhead is negligible (Table 7) — this represents the kernel-space implementation with Linux Kernel Modules. In the Internet case study, Decision Tree stills represent a good choice; however, other modules are also candidates for implementation, as is the Logistic Regression case. We see a more significant overhead compared to the first case study, and one of the reasons for this is the shifting of the implementation from kernel-space to userspace.

As future work, we plan to conduct a multi-label classification (narrowing the scope) to understand which reconnaissance activities are challenging to identify; improve the generation of datasets with more malicious cases; produce protection modules for the lwIP (lightweight IP), and test it in smaller operating systems such as ZephyrOS and FreeRTOS. Power consumption is a crucial performance metric to battery-aware systems, and we will include it in our future implementations.

A. OPEN SOURCE FRAMEWORK

The AB-TRAP framework, which allows to replicate the results shown in this paper and includes the source code for both kernel-space, and user-space applications, are available at https://github.com/c2dc/AB-TRAP/.

REFERENCES

[1] Rob Van Kranenburg and Alex Bassi. Iot challenges. Communications in Mobile Computing, 1(1):1–5, 2012.

[2] Francesca Meneghello, Matteo Calore, Daniel Zucchetto, Michele Polese, and Andrea Zanella. Iot: Internet of threats? a survey of practical security vulnerabilities in real iot devices. IEEE Internet of Things Journal, 6(5):8182–8201, 2019.

[3] Vikas Hassija, Vinay Chamola, Vikas Saxena, Divyansh Jain, Pranav Goyal, and Biplab Sikdar. A survey on iot security: application areas, security threats, and solution architectures. IEEE Access, 7:82721–82743, 2019.

[4] Abdulmalik Humayed, Jingqiang Lin, Fengjun Li, and Bo Luo. Cyber-physical systems security—a survey. IEEE Internet of Things Journal, 4(6):1802–1831, 2017.

[5] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, et al. The matter of heartbleed. In Proceedings of the 2014 conference on internet measurement conference, pages 475–488, 2014.

[6] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In 2010 IEEE symposium on security and privacy, pages 305–316. IEEE, 2010.

[7] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecurity, 2(1):20, 2019.

[8] A. L. Buczak and E. Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys Tutorials, 18(2):1153–1176, 2016.

[9] Preeti Mishra, Vijay Varadharajan, Uday Tupakula, and Emmanuel S Pilli. A detailed investigation and analysis of using machine learning techniques for intrusion detection. IEEE Communications Surveys & Tutorials, 21(1):686–728, 2018.

[10] Yang Xin, Lingshuang Kong, Zhi Liu, Yuling Chen, Yanmiao Li, Hongliang Zhu, Mingcheng Gao, Haixia Hou, and Chunhua Wang. Machine learning and deep learning methods for cybersecurity. IEEE Access, 6:35365–35381, 2018.

[11] Hongyu Liu and Bo Lang. Machine learning and deep learning methods for intrusion detection systems: A survey. Applied Sciences, 9(20):4396, 2019.

[12] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. A survey of network-based intrusion detection data sets. Computers & Security, 86:147–167, 2019.

[13] 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.

[14] Hanan Hindy, David Brosset, Ethan Bayne, Amar Seeam, Christos Tachtatzis, Robert Atkinson, and Xavier Bellekens. A taxonomy of network threats and the effect of current datasets on intrusion detection systems. IEEE Access, 2020.

[15] Monowar H Bhuyan, Dhruba Kumar Bhattacharyya, and Jugal K Kalita. Network anomaly detection: methods, systems and tools. Ieee communications surveys & tutorials, 16(1):303–336, 2013.

[16] Luca Arnaboldi and Charles Morisset. A review of intrusion detection systems and their evaluation in the iot. arXiv preprint arXiv:2105.08096, 2021.

[17] Maede Zolanvari, Marcio A Teixeira, Lav Gupta, Khaled M Khan, and Raj Jain. Machine learning-based network vulnerability analysis of industrial internet of things. IEEE Internet of Things Journal, 6(4):6822–6834, 2019.

[18] E. Viegas, A. O. Santin, and V. Abreu Jr. Machine learning intrusion detection in big data era: A multi-objective approach for longer model lifespans. IEEE Transactions on Network Science and Engineering, 8(1):366–376, 2021.

[19] Eduardo K Viegas, Altair O Santin, and Luiz S Oliveira. Toward a reliable anomaly-based intrusion detection in real-world environments. Computer Networks, 127:200–216, 2017.

[20] Giampaolo Bovenzi, Giuseppe Aceto, Domenico Ciuonzo, Valerio Persico, and Antonio Pescapé. A hierarchical hybrid intrusion detection approach in iot scenarios. 2020.

[21] 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.

[22] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: an ensemble of autoencoders for online network intrusion detection. arXiv preprint arXiv:1802.09089, 2018.

[23] Marek Małowidzki, P Berezinski, and Michał Mazur. Network intrusion detection: Half a kingdom for a good dataset. In Proceedings of NATO STO SAS-139 Workshop, Portugal, 2015.

[24] Z. Berkay Celik, Jayaram Raghuram, George Kesidis, and David J. Miller. Salting public traces with attack traffic to test flow classifiers. In Proceedings of the 4th Conference on Cyber Security Experimentation and Test, CSET'11, page 3, USA, 2011. USENIX Association.

[25] M. Zolanvari, M. A. Teixeira, and R. Jain. Effect of imbalanced datasets on security of industrial iot using machine learning. In 2018 IEEE International Conference on Intelligence and Security Informatics (ISI), pages 112–117, 2018.

[26] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. Mawilab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In ACM CoNEXT '10, CoNEXT '10, New York, NY, USA, 2010. Association for Computing Machinery.

[27] G. Karatas, O. Demir, and O. K. Sahingoz. Increasing the performance of machine learning-based idss on an imbalanced and up-to-date dataset. IEEE Access, 8:32150–32162, 2020.

[28] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In ICISSp, pages 108–116, 2018.

[29] H. He and E. A. Garcia. Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering, 21(9):1263–1284, 2009.

[30] Ross Anderson. Security engineering: a guide to building dependable distributed systems, chapter 21, page 723. John Wiley & Sons, 3rd edition, 2020.

[31] Wenliang Du. Computer Security: a hands-on approach. CreateSpace, 2017.

[32] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges. IEEE Transactions on Network and Service Management, 16(2):445–458, 2019.

[33] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In 2017 IEEE International Conference on Intelligence and Security Informatics (ISI), pages 43–48. IEEE, 2017.

[34] Geoffrey I Webb, Loong Kuan Lee, François Petitjean, and Bart Goethals. Understanding concept drift. arXiv preprint arXiv:1704.00362, 2017.

[35] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. Zero trust architecture. Technical report, National Institute of Standards and Technology, 2019.

[36] Pete Warden and Daniel Situnayake. Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers. " O'Reilly Media, Inc.", 2019.

[37] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Zmap: Fast internet-wide scanning and its security applications. In 22nd {USENIX} Security Symposium ({USENIX} Security 13), pages 605–620, 2013.

[38] Robert David Graham. Masscan: Mass ip port scanner. URL: https://github. com/robertdavidgraham/masscan, 2014.

[39] Eduardo Viegas, Altair O Santin, Andre Franca, Ricardo Jasinski, Volnei A Pedroni, and Luiz S Oliveira. Towards an energy-efficient anomaly-based intrusion detection engine for embedded systems. IEEE Transactions on Computers, 66(1):163–177, 2016.

[40] Christoph Molnar. Interpretable Machine Learning. Lulu. com, 2020.

[41] Tri Gia Nguyen, Trung V Phan, Binh T Nguyen, Chakchai So-In, Zubair Ahmed Baig, and Surasak Sanguanpong. Search: A collaborative and intelligent nids architecture for sdn-based cloud iot networks. IEEE access, 7:107678–107694, 2019.

[42] Gustavo de Carvalho Bertoli, Lourenço A Pereira, Cesar Marcondes, and Osamu Saotome. Evaluation of netfilter and ebpf/xdp to filter tcp flag-based probing attacks. In Proceedings of XXII symposium on operational applications in defense area (SIGE), pages 35–39, 2020.

[43] Eduard Glatz and Xenofontas Dimitropoulos. Classifying internet one-way traffic. In Proceedings of the 2012 Internet Measurement Conference, pages 37–50, 2012.

[44] Jon Nordby. emlearn: Machine Learning inference engine for Microcontrollers and Embedded Devices, March 2019.

[45] Ajay Tirumala. Iperf: The tcp/udp bandwidth measurement tool. http://dast.nlanr.net/Projects/Iperf/, 1999.

[46] Sebastien Godard. Sysstat utilities home page. Information and code available at http://sebastien.godard. pagesperso-orange.fr/index.html, 2015.

GUSTAVO DE CARVALHO BERTOLI received his BSc degree in Electrical Engineering (2012) by Universidade Estadual Paulista (UNESP), holds a MEng. in Mechanical-Aeronautical Engineering (2015) by the Technological Institute of Aeronautics – ITA. Currently, he is pursuing his PhD in Electronic Devices and Systems at Technological Institute of Aeronautics – ITA. His research interests include embedded systems, cyber security, machine learning, and internet of things.

LOURENÇO ALVES PEREIRA JÚNIOR received the B.Sc. degree (Hons.) in computer science from the University of Alfenas (UNIFENAS) in 2006 and the M.Sc. and Ph.D. degrees in computer science and computational mathematics from the University of Sao Paulo (ICMC/USP), 2010 and 2016, respectively. He is currently an Assistant Professor with the Department of Computer Systems at the Brazilian Aeronautics Institute of Technology—ITA. He is also a member of the Laboratory of Command and Control and Cyber Security. His main research interests include computer networks and cybersecurity. Lourenço has also acted as a reviewer for IEEE ComMag, IEEE ACCESS, IEEE LATAMT, IEEE TPDS, JNSM, Elsevier Vehicular Communications.

OSAMU SAOTOME received his BSc degree in Electronic Engineering (1974) by Technological Institute of Aeronautics – ITA. He held Master's degree from Institute of Technology of Tokyo, Japan (1984) and also, he held a PhD of Engineering Degree (1987) from Institute of Technology of Tokyo. Currently, he is an Associate Professor Technological Institute of Aeronautics. He has experience in Electronic Engineering, focused on Electronic Circuits, Microprocessors, Digital Signal Processors (DSPs), Embedded Electronic Systems, Field Programmable Gate Arrays (FPGAs), Intelligent Sensors, Embedded Software, in subjects such as real-time systems, Electronic Systems and Devices, Digital Signal Processing Algorithms, Digital Image and Video Processing. He also has experience in Mechatronic Systems Engineering, at the major of Aerial and Space Robotics, focusing on Embedded Systems of Quadrotors and Collaborative Fleet Formation of Quadrotors.

ALDRI L. DOS SANTOS is professor of the Department of Computer Science at Federal University of Minas Gerais (UFMG). Aldri is PhD in Computer Science from the Federal University of Minas Gerais, Master in Informatics and Bachelor of Computer Science at UFPR. Aldri is working in the following research areas: network management, fault tolerance, security, data dissemination, wireless ad hoc networks and sensor networks. He is leader of the research group (Wireless and Advanced Networks). Aldri has also acted as reviewer for publications as IEEE ComMag, IEEE ComNet, ComCom, IEEE Communications Surveys and Tutorials, IEEE eTNSM, JNSM, Ad hoc Networks. Aldri has served as member of the technical committee of security information and IEEE Communication Society Communication (ComSoc).

FILIPE ALVES NETO VERRI received the B.Sc. degree (Hons.) in computer science and the Ph.D. degree (Hons.) in computer science and computational mathematics from the University of São Paulo, in 2014 and 2018, respectively. In 2016, he was a Visiting Scholar with the School of Electrical, Computer and Energy Engineering, Arizona State University, under the supervision of Prof. Ying-Cheng Lai. He is currently an Assistant Professor with the Division of Computer Science, Department of Computer Methods, Instituto Tecnológico de Aeronáutica (ITA). His research interests include data science, machine learning, complex networks, and complex systems.

CESAR AUGUSTO CAVALHEIRO MARCONDES holds a PhD in Computer Science (2008) from UCLA. He is an assistant professor in the Department of Computer Systems, ITA. His main research interests are computer networks and security. From 2008 to 2018, he was Assistant Professor with Federal University of Sao Carlos – UFSCar working on Future Internet and Congestion Control. He also holds industrial patents from USPTO in congestion control while working at Sun Microsystems.

SIDNEI BARBIERI received his BSc degree in Computer Science (2009) by Regional Integrated University (URI) of Santo Ângelo and holds MSc in Computer Science (2018) by Aeronautics Institute of Technology (ITA). He is officer in the Brazilian Army's Cyber Defense Center and works with research and analysis of cyber threats. His research interests are Data Science and Computer Networks.

MOISÉS S. RODRIGUES received his Bsc degree in Computer Engineering (2007) by Military Institute of Engineering (IME) and holds Msc in Electronics and Computer Engineering by Aeronautics Institute of Technology (ITA). He works with Information Security and IT Services Management at Brazilian Army. His research interests includes Computer Networks and Cyber Security.

JOSE M. PARENTE DE OLIVEIRA became a Member (M) of IEEE in 2017. He received a M.S. (1996) and a Ph.D (2003) degrees in Electronics and Computing Engineering from the Aeronautics Institute of Technology (ITA), Brazil. His research interests include ontology, big data, data science and semantic web. He is a full professor at the Computer Science Department at ITA and a collaborating researcher at The National Center for Ontological Research (NCOR) and at the Center for Multipurpose Information Fusion (CMIF), in The State University of New York – University at Buffalo. Dr. Parente de Oliveira has been responsible for projects related to the use of ontologies, web services and big data in the context of Air Traffic Management in Brazil. He was also in charge of administrative positions at ITA, such as the Head of the Computer Science Department and Graduate Studies Sector.