The Cross-evaluation of Machine Learning-based Network Intrusion Detection Systems

Apruzzese, Giovanni; Pajola, Luca; Conti, Mauro

DOI
10.1109/TNSM.2022.3157344

Publication date
2022

Document Version
Final published version

Published in
IEEE Transactions on Network and Service Management

The Cross-Evaluation of Machine Learning-Based Network Intrusion Detection Systems

Giovanni Apruzzese, Luca Pajola, and Mauro Conti, *Fellow, IEEE*

***Abstract*—Enhancing Network Intrusion Detection Systems (NIDS) with supervised Machine Learning (ML) is tough. ML-NIDS must be trained and evaluated, operations requiring data where benign and malicious samples are clearly labeled. Such labels demand costly expert knowledge, resulting in a lack of real deployments, as well as on papers always relying on the same outdated data. The situation improved recently, as some efforts disclosed their labeled datasets. However, most past works used such datasets just as a 'yet another' testbed, overlooking the added potential provided by such availability.**

In contrast, we promote using such existing labeled data to *cross-evaluate* ML-NIDS. Such approach received only limited attention and, due to its complexity, requires a dedicated treatment. We hence propose the first cross-evaluation model. Our model highlights the broader range of realistic use-cases that can be assessed via cross-evaluations, allowing the discovery of still unknown qualities of state-of-the-art ML-NIDS. For instance, their detection surface can be extended—at no additional labeling cost. However, conducting such cross-evaluations is challenging. Hence, we propose the first framework, XeNIDS, for reliable cross-evaluations based on Network Flows. By using XeNIDS on six well-known datasets, we demonstrate the concealed potential, but also the risks, of cross-evaluations of ML-NIDS.

***Index Terms*—Machine learning, intrusion detection systems, network security, evaluation.**

I. Introduction

MACHINE Learning (ML) is advancing at a rapid pace (e.g., [1], [2]), and the cybersecurity domain is also looking at ML with great interest [3]. ML methods can automatically learn to make decisions by using existing data, representing a valuable asset to monitor the increasingly mutating IT environments.

Although ML is already deployed to counter some threats (e.g., malware or phishing [4], [5], [6]), ML methods are still at an early stage for Network Intrusion Detection (NID). In particular, some Network Intrusion Detection Systems (NIDS) integrate commercial products that use *unsupervised* ML (e.g., [7], [8]). Such solutions can be useful to perform correlation analyses or to 'detect anomalies,' which are ancillary to true intrusion detection tasks (an anomaly is not necessarily an intrusion). The full potential of ML can be appreciated only via *supervised* methods, which assume the existence of *labels* that associate each sample to its ground truth [9]. Specifically in NID, by creating a training dataset where the samples are distinguished between *benign* and *malicious*, it is possible to develop a fully autonomous Machine Learning-based Network Intrusion Detection System (ML-NIDS).

Deployment of ML-NIDS involves two stages: the system must first be developed (i.e., *trained*), and it must then be *evaluated*, because any security system that has not been tested is dangerous [10]. Both of these stages require *large* amounts of *labeled* data, which can only be collected via the supervision of a human that associates (and verifies) each sample to its ground truth [11]. While such verifications are simple in some applications (e.g., any layman can distinguish a legitimate from a phishing website), the inspection of network data requires expert knowledge–which is expensive [12]. To aggravate the problem, a network can be targeted by many attacks, *each* of which must be labeled to assess the detection capabilities of a ML-NIDS. As a result, the inevitable and costly necessity of comprehensive labeled datasets (usually numbering millions of samples [13]) discourages deployment of ML-NIDS. We note that this problem also extends to *research*. For more than a decade, the only publicly available dataset for ML-NIDS was the KDD99, leading to a plethora of works always trained and evaluated on such dataset—usually with perfect performance (e.g., [14]).

To address the lack of labeled data, recent researches on ML-NIDS openly released their datasets (e.g., [13], [15], [16]), an effort appreciated by related literature (e.g., [14], [17], [18]). However, most prior works used such datasets as an additional testbed for their proposals. As a result, such works only confirmed what was already known: that by training a ML-NIDS on a (large) dataset, such ML-NIDS will detect the attacks contained in such dataset. This is because the primary objective was to 'outperform' the state-of-the-art, resulting in incremental contributions that do not foster realistic deployments. In this paper, we aim to broaden such limited scope.

Inspired by a recent paper by Pontes *et al.* [19], we observe that the current availability of labeled datasets could be better exploited by ML-NIDS researches. Specifically, we endorse the idea of *cross-evaluating* ML-NIDS by using malicious

Manuscript received 8 October 2021; revised 4 January 2022; accepted 1 March 2022. Date of publication 8 March 2022; date of current version 31 January 2023. This work was supported by the European Commission under the Horizon 2020 Programme (H2020), as part of the LOCARD Project under Grant 832735. The associate editor coordinating the review of this article and approving it for publication was J.-H. Cho. *(Corresponding author: Giovanni Apruzzese.)*

Giovanni Apruzzese is with the Institute of Information Systems, University of Liechtenstein, 9490 Vaduz, Liechtenstein (e-mail: giovanni.apruzzese@uni.li).

Luca Pajola is with the Department of Mathematics, University of Padua, 35122 Padova, Italy (e-mail: pajola@math.unipd.it).

Mauro Conti is with the Department of Mathematics, University of Padua, 35122 Padova, Italy, and also with the Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, 2628 CD Delft, The Netherlands (e-mail: conti@math.unipd.it).

Digital Object Identifier 10.1109/TNSM.2022.3157344

samples captured in different network datasets.[1] By performing such cross-evaluations, it is possible to gauge additional properties of ML-NIDS, allowing a better understanding of the state-of-the-art at no extra labeling cost.

To the best of our knowledge, this is the first effort that focuses on the opportunity provided by cross-evaluations of ML-NIDS. As such, our primary goal is the definition of a data-agnostic *model* that allows to represent such cross-evaluations. Indeed, using samples from different networks is not straightforward: as stated by Sommer and Paxson [9], each network has "immense variability", suggesting that cross-evaluations have intrinsic risks that must be known to avoid deployments of unreliable ML-NIDS. Our model acknowledges such risks, but also highlights the *benefits* that can be brought by cross-evaluations of ML-NIDS. Such benefits come in the form of additional types of 'contexts' that can be reproduced in research environments, each representing a distinct realistic use-case. Specifically, our model highlights the limited scope of the state-of-the-art, whose fixed evaluation methodology can only cover 2 contexts, whereas cross-evaluations can span over up to 10 different contexts. Such broad range evidences the concealed potential of the core idea at the base of our paper.

As stated by Biggio *et al.* [10], ML systems for cybersecurity must be assessed *in advance*. Therefore, *proactive* cross-evaluations must take into account all the pitfalls highlighted by our model. To further promote our proposal, we develop the first framework for cross-evaluations of ML-NIDS, XeNIDS. Our framework aims to overcome the intrinsic challenges of cross-evaluations, while allowing the reproduction of all contexts enabled by our model. Specifically, XeNIDS focuses on NetFlow data, which is popular in the ML-NIDS community (e.g., [13], [19], [21]). However, using NetFlows from different environments is tough: such data can be generated in many ways, resulting in heterogeneous formats that may lead to unreliable ML-NIDS. We address this issue via an *original* interpretation of NetFlows w.r.t. ML. Using this interpretation, we provide the guidelines that can increase the reliability of the results provided by XeNIDS.

As an instructive *demonstration*, we use XeNIDS to perform a large cross-evaluation of ML-NIDS spanning over 6 well-known and recent datasets. We aim to reproduce realistic use cases, which can be assessed via three different context types enabled by our model. Specifically, we first consider the 'baseline' context commonly adopted by prior work, and show that XeNIDS yields the same performance as the state-of-the-art. Then, we assess the context where a ML-NIDS is *tested* on malicious samples originating from different networks; such use-case was also investigated in [19], and XeNIDS matches their performance. Finally, we assess the context where the ML-NIDS is *trained and tested* on malicious samples from different networks, showing a dramatic performance increase. As a final contribution of this paper, we provide an in-depth *analysis* of these results, where we investigate their reliability for practical deployments.

[1]We stress that our term 'cross-evaluation' denotes a different concept than the term 'cross-validation' commonly used in ML researches [20].

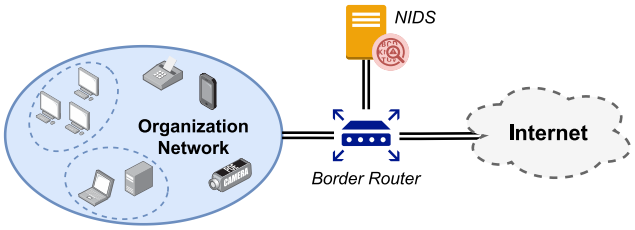


Fig. 1. Typical NIDS scenario. The network of the organization can be composed of multiple subnetworks. The outgoing traffic passes through a border router which forwards such traffic to the internet, but also to a NIDS. The NIDS analyzes the traffic and, if necessary, raises some alerts.

Contribution and Organization. This is the first paper that addresses the problem of cross-evaluations of ML-NIDS. As such, the specific contributions are as follows.

- We present the first data-agnostic model that conceptualizes the problem of cross-evaluation of ML-NIDS.
- We use our model to showcase the benefits and challenges of such cross-evaluations.
- We propose XeNIDS, the first framework for reliable cross-evaluations focused on NetFlow data.
- We demonstrate all of the above by cross-evaluating ML-NIDS over 6 distinct well-known datasets, and analyzing the results' reliability.

The remainder of the paper has the following structure. We motivate our paper in Section II. We define our cross-evaluation model in Section III. We describe our XeNIDS framework in Section IV. We explain the application of XeNIDS in Section V. We present our demonstration in Section VI. We discuss the results in Section VII. We conclude our paper in Section VIII.

II. Background

This work lies at the intersection of Machine Learning and Network Intrusion Detection. We first provide some preliminary information on these two areas (Section II-A). Then, we explain the motivation (Section II-B) of our paper. Finally, we compare this effort with related work (Section II-C).

A. Network Intrusion Detection and Machine Learning

The so-called security lifecycle spans over three activities: prevention, detection, reaction [22]. However, the prevention of any cyber-attack is an impossible task, while the reaction phase assumes that most of the damage has already taken place. For this reason, proposals focusing on the detection step have received much more attention, as timely and accurate identifications of cyber threats can significantly mitigate the effects of an offensive campaign [23].

In the specific domain of network security (which is of interest to this paper), the detection of such malicious events is devoted to Network Intrusion Detection Systems (NIDS). We provide a schematic representation of the typical NIDS deployment in Fig. 1, where a NIDS inspects the traffic generated by the monitored network (and all of its subnetworks). A NIDS can leverage two distinct detection paradigms, which are based either on *fixed rules* or on *data-driven* methods [24]. The former requires human operators that write specific rules

(or signatures) that denote a specific threat, and exhibit high performance against *known* and *static* threats whose behavior is captured by the hardcoded rules. On the other hand, the latter leverage automatic data analyses and can detect even *unknown* and *mutating* threats if they present similarities with previously known samples–potentially at the expense of higher false-positive rates.

The increased growth of data alongside improvements in collaborative computing resulted in a huge interest in data-driven NIDS, specifically employing machine learning methods [3], [11]. Such methods involve a *training* phase where the ML model learns to make decisions from existing data. However, without some reference information, it is not possible to control what the ML model is actually 'learning' [25]. To specifically address detection (i.e., classification) problems, the training data must be separated into *benign* and *malicious* samples. In such circumstances, it is possible to develop autonomous ML-NIDS exploiting supervised ML methods. Such 'supervision' comes in the form of a human that must associate each sample in the training data to its ground truth, i.e., a *label* [9].

In some domains, labeling is simple (e.g., the popular captchas [26]) or labeled data can be used for a long period of time (e.g., ImageNet was collected in 2009 and is still widely used today [27]). However, the Cybersecurity domain is different: according to Miller *et al.* [12], a company can only label 80 malware samples per day. Specifically in NID, ground truth verification of network data is complex [28], and the concept drift problem requires any ML-NIDS to be continuously updated with new–labeled–data [29]. To aggravate this problem, deployment of any security system requires *proactive* evaluations conducted *in advance*, to avoid introducing a weak link in the security chain [10]. Hence, in the case of ML-NIDS, labeled data must be obtained both for the initial training, as well as for such evaluation.

The scarcity of labeled data for NID affected both research and practice [30], with an overall lack of ML-NIDS deployments, as well as a plethora of papers always based on the only publicly available dataset—the KDD99 [14].

B. Motivation: Mixing Network Data

The successes of ML renewed the interest of the NID community in these methods, and in recent years, many labeled datasets were made openly accessible (a survey is in [13]). However, most related work simply used such data as an 'additional' setting to perform their experiments. In contrast, in this paper we promote a different approach, based on mixing different network data to cross-evaluate ML-NIDS. Such opportunity, fostered by the recent availability of NID datasets, is of interest both for research and practice. Let us explain how mixing network data can assist ML-NIDS deployment. We first by present some high level applications (Section II-B1), and then provide a more specific use-case (Section II-B2).

1) Applications and Advantages: Mixing data from different networks is useful to augment pre-existing datasets that contain an insufficient amount of labeled samples to develop ML-NIDS. It is also useful to assess the generalization capabilities of a ML-NIDS against 'novel' attacks not included in the training set (as very recently done by [19]). Such 'novel' samples can also be used extend the detection surface of the ML-NIDS by injecting them in the training set of the ML-NIDS. Similar strategies are particularly relevant to protect against the so-called 'adversarial attacks' which can evade traditional ML-NIDS [31]: the (new) training data can be leveraged for *adversarial training*, therefore realizing robust ML systems that can detect even subtle perturbations [32]. In this context, mixing diverse datasets facilitates the application of *ensemble* techniques (e.g., [33]), further increasing the resilience of ML-NIDS.

As stated by Biggio and Roli, empirical evaluations are always necessary for real deployments [34]. In this context, cross-evaluations are advantageous due to their low opportunity cost—especially when using publicly available data. Indeed, we observe that great attention has been given to *data sharing* platforms (e.g., [35]), and cross-evaluations could greatly benefit from dedicated 'banks' of NID data (e.g., [36]): it is true that the cybersecurity domain has high confidentiality, but anonymization techniques exists [37], and some recent solutions in *federated learning* overcame privacy issues (e.g., [38]). Finally, cross-evaluations can involve even *unsupervised* ML methods (e.g., anomaly detectors [39]), which represent the majority of currently deployed ML techniques for NIDS. Although unsupervised methods would not benefit from the 'cheap' labeling, they can still take advantage of the data diversity of different networks to assess (or improve) their generalization capabilities.

2) Exemplary Use-Case: Suppose an organization, $\mathcal{O}$, wants to protect their network, o, with a (supervised) ML-NIDS. Hence, $\mathcal{O}$ collects and verifies some *benign* traffic data, N, from their network o. However, ML-NIDS also require *malicious* data, M. The following can happen w.r.t. such M:

- $\mathcal{O}$ may not have any M generated in their network o. Hence, $\mathcal{O}$ can 'use' some M generated in a different network than o – potentially of another organization.
- $\mathcal{O}$ may have some M generated in o, obtained, e.g., by monitoring the behavior of 'known' infected machines.

Therefore, $\mathcal{O}$ can use such N and M to develop any ML model which, if it obtains appreciable performance, will be integrated in their security system as a ML-NIDS that can detect the attacks in M. Having an operational ML-NIDS, $\mathcal{O}$ may be willing to assess whether such system can detect attacks not included in their M, which can potentially target the network o monitored by the ML-NIDS. To this end, $\mathcal{O}$ can use a *small* set of malicious data originating from a network different than o, and containing different attacks than the ones 'learned' by their ML-NIDS. By using such malicious data to *evaluate* the ML-NIDS, $\mathcal{O}$ can assess the generalization capabilities of their solution. If the assessment shows a weakness of the ML-NIDS, $\mathcal{O}$ may acquire a *larger* set of such malicious data to extend the detection capabilities of their ML-NIDS, by using such data in the *training* stage. We will use the abovementioned example as basis for our demonstration in Section VI.

C. Related Work

The idea of cross-evaluating ML-NIDS on different datasets is not new. For instance, the authors of [40] propose a novel

IDS dataset that can be used to evaluate the 'transferability' of ML-NIDS, but they do not provide any detailed analysis nor original experiment. Similarly, Pontes *et al.* [19] use a ML-NIDS trained on `IDS18` against `DDoS19`. However, [19] simply limit to *test* a novel method on a different dataset, and do not analyze the problem of 'cross-evaluations' as a whole, hence not allowing to highlight the benefits and limitations of such opportunity. For instance, cross-evaluations can also involve modifications of the *training* data, which is not covered by [19] and which is a case included in our demonstration.

Most prior works on ML-NIDS only assess their proposals in a single 'context', that is, the training and evaluation use the same dataset. For instance, the authors of [41] propose botnet detectors trained and tested on `CTU13`. In [42] a ML-NIDS focusing on different attacks is assessed on the `IDS18` dataset. To give a practical explanation, such methodology only allows determining that "the approach in [42] is effective on the network captured by the `IDS18` dataset, against the attacks contained in the `IDS18` dataset". Other works may consider more datasets (e.g., [18], [21], [43], [44]), but the problem remains because the assessments are carried out independently on each dataset. Furthermore, all these works highlight that ML-NIDS require large (labeled) datasets–further motivating the need to explore novel solutions that mitigate the lack of labeled data. Among these, we mention *semisupervised* ML approaches (e.g., [28], [45]), which combine unlabeled with labelled data, and are hence orthogonal to our work.

A closely related research effort is [46], proposing a low-level software toolkit for analyzing NID datasets, forcing the user to abide to its constrained logic. For instance, it only works with data in the form of packet captures (PCAP), which require huge amounts of storage space and whose payload is often encrypted, making such data impractical to share (and, also, to analyze). In contrast, our proposed model is agnostic of the source data format (as long as there is some compatibility); moreover, our proposed framework operates on Network Flows (NetFlows), which represent a higher level than PCAP, making it flexible and extendible also to PCAP data—while not sacrificing performance [17], [41], [47].

We conclude that the idea of cross-evaluating ML-NIDS received only limited attention so far, and its opportunities and risks are still unknown. This is because no past research truly addressed such a problem—representing the core of this paper. Our intention is to provide a complete understanding of all the pros and cons related to cross-evaluations of ML-NIDS.

III. Modeling the Cross-Evaluation of ML-NIDS

The intuition at the base of our work is to leverage *existing* NID datasets, with the goal of cross-evaluating ML-NIDS using samples from mixed networks. Such idea is grounded on the following observation (also implicitly adopted by [19]), which extends the takeaways by Sommer et Paxson [9]: although every network is unique, the malicious behavior of network attacks is independent of the target network. For instance, Denial of Service (DoS) attacks always involve either a large amount of communications with minimal entity, or a smaller set of communications but with a larger entity – both happening in a short time frame [48]. Similarly, a machine infected by Botnet malware will periodically contact the CnC server, irrespective of what is happening in the 'compromised' network [49]. Hence, such malicious behaviors can be used by a ML-NIDS to distinguish benign from malicious activities, regardless of the target network.

As the first effort to investigate this opportunity, we must design a *model* (Section III-A) that allows to highlight its *benefits* (Section III-B) as well as its intrinsic *challenges and risks* (Section III-C).

A. Proposed Model Design

We now introduce all the prerequisites to describe our cross-evaluation model.

Let $\mathbb{D}$ be a set of NID datasets which we denote as follows:

$$\mathbb{D} = (D_1, D_2, \ldots, D_n), n \geq 2,$$

where n represents the cardinality of $\mathbb{D}$, and D_i represents a dataset collected in a given network i. Without loss of generality, we assume that each $D_i \in \mathbb{D}$ originates from a unique network environment—potentially, $\mathbb{D}$ can include datasets representing distinct sub-networks within a larger network. Hence, in the remainder we use D_i and D_j $(i \neq j)$ to denote two datasets of $\mathbb{D}$ originating from two distinct networks (i.e., i and j). The *information* captured by each dataset in $\mathbb{D}$ must allow one to use any subset of $\mathbb{D}$ and derive a *set of common features* from such subset.[2]

Because our focus is on supervised ML for detection problems, each $D_i \in \mathbb{D}$ must be provided with *ground truth* distinguishing benign from malicious data. Hence, each dataset D_i can be seen as a composition of N_i, denoting the *benign* data of network i, and M_i, denoting the *malicious* data of network i. A dataset D_i can contain only malicious (or only benign) data; however, across all $\mathbb{D}$ there must be at least a pair D_i, D_j for which $N_i \neq \varnothing$ and $M_j \neq \varnothing$. We denote with μ the number of *all* malicious classes contained in the entire $\mathbb{D}$. This is because any D_i can have a M_i with a variable number of attacks (i.e., Botnet, DoS, etc), which may overlap (or not) with those in a different M_j. Therefore, every M_i can be seen as an array of μ elements $M_i = (M_i^1, M_i^2, \ldots, M_i^\mu)$, some of which can be empty if another dataset D_j has malicious classes not contained in D_i. Let $\mathbb{N}$ denote the set of all benign samples, and $\mathbb{M}$ denote the set of all malicious samples.

We can visualize our model with the schematic in Fig. 2, which shows the relationship between $\mathbb{D}$, $\mathbb{N}$ and $\mathbb{M}$.

From Fig. 2, we observe that all sets have n rows, each denoting a distinct source network dataset. However, while $\mathbb{N}$ has only one column because all benign samples are treated equally, $\mathbb{M}$ has μ columns representing all the attacks contained in $\mathbb{D}$. We provide an example in the caption of Fig. 2.

Such design makes our model suitable for $1 + \mu$ classification ML problems, where a sample is either benign, or belongs

[2]For instance, it is possible that D_i comes as PCAP traces, and D_j comes as NetFlows: in this case, the PCAP of D_i can be processed to derive the NetFlows features of D_j. Similarly, two datasets D_i and D_j can contain NetFlows generated with different software: in this case, the features shared by D_i and D_j can represent the common set.

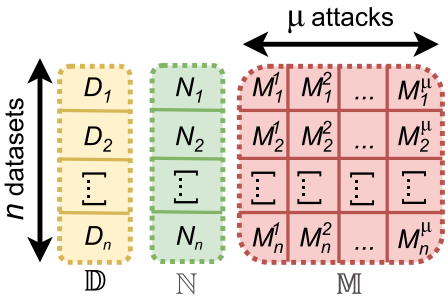


Fig. 2. Proposed cross-evaluation model, representing $\mathbb{D}$, $\mathbb{N}$ and $\mathbb{M}$. **Example**: assume that $\mathbb{D}$ has three datasets: D_1, D_2, D_3 (hence $n = 3$). Assume that: D_1 has some benign samples but no malicious samples; D_2 has no benign samples, but malicious samples of a *botnet* attack, and also some malicious samples of a *portscan* attack; D_3 has some benign samples, alongside some malicious samples of the same *botnet* attack as D_2, but also some malicious samples of a *DoS* attack. In this case, $\mathbb{N}$ will contain three elements: N_1, N_2, N_3, with N_2 being empty. On the other hand, there are three malicious classes ($\mu = 3$) hence $\mathbb{M}$ will have nine elements: M_2^1 and M_3^1 (the botnet attack shared by D_2 and D_3) as well as M_2^2 (the portscan attack in D_2) and M_3^3 (the DoS attack in D_3) will contain some samples; whereas the remaining will be empty (i.e., all those with M_1, as well as M_2^3 and M_3^2).

to one among μ malicious classes. This automatically covers binary classification ML problems if all malicious classes are treated as a single malicious class (hence, $\mu = 1$).

Let us now use our proposed model to explain the benefits brought by cross-evaluations of ML-NIDS.

B. Benefits: Additional Contexts

Deployment of ML components requires a *training* set T, used to develop a ML model, and an *evaluation* set E, used to assess the performance of such model. Hence, our idea is composing T and E by drawing from $\mathbb{N}$ and $\mathbb{M}$: depending on the draw, a specific 'context' is created that can be used to cross-evaluate a ML-NIDS. The main benefits provided cross-evaluations of ML-NIDS are due to the increased types of contexts that can be assessed, which can be highlighted with our proposed model.

Because our model is rooted on Sommer and Paxson statement (Section III), it is crucial that both T and E use benign samples from the same network[3], which should represent the environment where the ML-NIDS is to be deployed; hence, let o (standing for 'origin') denote such network, and N_o be the corresponding benign samples. Then, we observe that there are many ways to compose T and E by choosing the malicious element from $\mathbb{M}$. Such variability can be modeled through the 'matrix' $\mathbb{M}$ in Fig. 2, by pinpointing which rows and columns are included in T and E. The following can occur:

- T (or E) can contain malicious samples either from the same or different o (i.e., same or different row than N_o);
- the malicious samples in T and E can come either from the same or different networks (regardless of N_o);
- the malicious class(es) in T can be either the same or different than those in E (i.e., same or different columns).

[3]This also serves to reduce false alarms *after* potential deployments, because the benign samples always have the same source.

In particular, let t and e denote two rows of $\mathbb{M}$; and let τ and ε denote two columns of $\mathbb{M}$; we use such notation to identify two elements of $\mathbb{M}$, i.e., M_t^τ and M_e^ε.

Let $\vec{t}$, $\vec{e}$, $\vec{\tau}$, $\vec{\varepsilon}$ be four *ordered arrays*[4], each denoting multiple columns or rows (e.g., $\vec{t}$ contains multiple t, i.e., rows) of $\mathbb{M}$. Let $\vec{o}$ be an unary array including only o (representing the benign 'origin' network from $\mathbb{N}$).

By following such notation, we can represent the training and evaluation sets, T and E, as the functions in Expression 1:

$$T(\vec{o}, \vec{t}, \vec{\tau}), \quad E(\vec{o}, \vec{e}, \vec{\varepsilon}). \tag{Exp. 1}$$

Simply put, T and E are denoted by a single row of $\mathbb{N}$ (i.e., $\vec{o}$), and the rows and columns of all the elements of $\mathbb{M}$ that they include (i.e., $\vec{t}$ and $\vec{\tau}$ for T, while $\vec{e}$ and $\vec{\varepsilon}$ for E).

We can see a *context* as a function of T and E. Specifically, a context C is denoted as the following tuple:

$$\mathsf{C}(T, E) \Rightarrow \mathsf{C}(\vec{o}, \vec{t}, \vec{e}, \vec{\tau}, \vec{\varepsilon}). \tag{Exp. 2}$$

Depending on the elements from $\mathbb{N}$ and $\mathbb{M}$ included in T and E, many contexts can be reproduced, which can be of different *type*. In particular, let $\bar{o}, \bar{h}, \bar{e}, \bar{\tau}, \bar{\varepsilon}$ denote the *sets* of the corresponding arrays (each element of a given set is unique). By cross-evaluating ML-NIDS, it is possible to assess 10 different context types, which are denoted by the relationships between $\bar{o}, \bar{h}, \bar{e}, \bar{\tau}, \bar{\varepsilon}$.

We provide the full list of such context types in Table I; we also include a practical example in the caption of Table I. Specifically, for each context type (denoted with a number after the letter C), we report the four conditions denoting the relationships among all the involved components; on the same line of each condition, we describe the consequences on T and E; we also provide a concrete use case that explains the application of such type of context. We note that all cases where two sets are not equal can be further split in two: when one set is a *superset* of the other; and when they are *disjointed*.

Past works (e.g., [17], [50]) only considered cases where the 'row' was fixed, i.e., where $\bar{o} = \bar{h} = \bar{k}$, corresponding to the contexts of type $\mathsf{C1}$ and $\mathsf{C2}$. Pontes *et al.* [19] investigated $\mathsf{C4}$. In contrast, it is evident from Table I that our cross-evaluation model enables the assessment of 7 additional context types, allowing to discern additional qualities of MLe-NIDS and corresponding NID datasets. For instance, all the scenarios envisioned in our motivational example (cf. Section II-B) can be represented by the context types listed in Table I.

In our demonstration (Section VI), we first assess $\mathsf{C1}$ as 'baseline' comparison with the state-of-the-art; and then we consider $\mathsf{C4}$ (as done by [19]), and $\mathsf{C7}$ (the latter both in its 'disjointed' and 'extended' variants).

C. Challenges and Risks

The cross-evaluation of ML-NIDS has high potential, but a superficial application can lead to dangerous consequences—spanning from underwhelming performance to

[4]Such arrays can be of variable length, as long as $|\vec{t}|=|\vec{\tau}|$ and $|\vec{e}|=|\vec{\varepsilon}|$.

TABLE I

THE 10 TYPES OF CONTEXTS ENABLED BY THE PROPOSED CROSS-EVALUATION MODEL. THE TABLE REPRESENTS THE RELATIONSHIP AMONG THE UNIQUE ELEMENTS OF $\mathbb{N}$ AND $\mathbb{M}$ INCLUDED IN THE TRAINING AND EVALUATION SETS, T AND E (CF. EXPR. 1), WHICH DENOTE A CONTEXT (CF. EXPR. 2). THE ORIGIN OF BENIGN SAMPLES IS ALWAYS THE SAME IN T AND E, HENCE $\bar{o}$ IS SHARED. CONTEXTS OF TYPE C1, C2 ARE THOSE CONSIDERED BY MOST PRIOR WORKS; C4 HAS BEEN CONSIDERED IN [19]. A GRAY BACKGROUND DENOTES THE TYPES OF CONTEXT ASSESSED IN OUR DEMONSTRATION. **EXAMPLE:** CONSIDER THE SETTING DESCRIBED IN THE EXAMPLE OF FIG. 2, WHERE $n = 3$ AND $\mu = 3$. SUPPOSE A CONTEXT DENOTED BY: $\vec{o} = (3), \vec{t} = (2, 3), \vec{e} = (3), \vec{\tau} = (1, 1), \vec{\varepsilon} = (1)$. SUCH CONTEXT IMPLIES THAT T CONTAINS BENIGN SAMPLES FROM N_3, AND MALICIOUS SAMPLES FROM M_2^1 AND M_3^1; WHEREAS E CONTAINS BENIGN SAMPLES FROM N_3 BUT MALICIOUS SAMPLES FROM M_3^1. THE RESULTING CONTEXT WILL BE OF TYPE C5, BECAUSE $\bar{o} = (3), \bar{t} = (2, 3), \bar{e} = (3), \bar{\tau} = (1), \bar{\varepsilon} = (1)$. HENCE, $(\bar{o} = \bar{e}) \neq \bar{t}$ AND $\bar{\tau} = \bar{\varepsilon}$

C-type	Conditions	Effects on T and E	Use-case
C1	$\bar{o} = \bar{t}$ $\bar{o} = \bar{e}$ $\bar{t} = \bar{e}$ $\bar{\tau} = \bar{\varepsilon}$	T uses benign and malicious samples from the same network. E uses benign and malicious samples from the same network. T and E use malicious samples from the same network. T and E use the same attack classes.	This is the standard 'baseline' case commonly used in research.
C2	$\bar{o} = \bar{t}$ $\bar{o} = \bar{e}$ $\bar{t} = \bar{e}$ $\bar{\tau} \neq \bar{\varepsilon}$	T uses benign and malicious samples from the same network. E uses benign and malicious samples from the same network. T and E use malicious samples from the same network. T and E use different attack classes.	Same as the baseline C1 but the ML-NIDS is tested on different attacks.
C3	$\bar{o} = \bar{t}$ $\bar{o} \neq \bar{e}$ $\bar{t} \neq \bar{e}$ $\bar{\tau} = \bar{\varepsilon}$	T uses benign and malicious samples from the same network. E uses benign and malicious samples from different networks. T and E use malicious samples from different networks. T and E use the same attack classes.	Using the ML-NIDS on the same attacks, but targeting hosts from a different network.
C4	$\bar{o} = \bar{t}$ $\bar{o} \neq \bar{e}$ $\bar{t} \neq \bar{e}$ $\bar{\tau} \neq \bar{\varepsilon}$	T uses benign and malicious samples from the same network. E uses benign and malicious samples from different networks. T and E use malicious samples from different networks. T and E use different attack classes.	Testing the generalization capabilities of the ML-NIDS on completely unknown attacks (different hosts, different attacks).
C5	$\bar{o} \neq \bar{t}$ $\bar{o} = \bar{e}$ $\bar{t} \neq \bar{e}$ $\bar{\tau} = \bar{\varepsilon}$	T uses benign and malicious samples from different networks. E uses benign and malicious samples from the same network. T and E use malicious samples from different networks. T and E use the same attack classes.	When there are too few samples of an attack class to both train and test a ML-NIDS, it is possible to borrow some malicious samples from another network and use them in T.
C6	$\bar{o} \neq \bar{t}$ $\bar{o} = \bar{e}$ $\bar{t} \neq \bar{e}$ $\bar{\tau} \neq \bar{\varepsilon}$	T uses benign and malicious samples from different networks. E uses benign and malicious samples from the same network. T and E use malicious samples from different networks. T and E use different attack classes.	Assessing whether training on new attacks (different hosts and attack class) affects the detection on 'known' attacks.
C7	$\bar{o} \neq \bar{t}$ $\bar{o} \neq \bar{e}$ $\bar{t} = \bar{e}$ $\bar{\tau} = \bar{\varepsilon}$	T uses benign and malicious samples from different networks. E uses benign and malicious samples from different networks. T and E use malicious samples from the same network. T and E use the same attack classes.	Extending the detection surface of the ML-NIDS by training on attacks originating from different networks and testing such ML-NIDS against these attacks.
C8	$\bar{o} \neq \bar{t}$ $\bar{o} \neq \bar{e}$ $\bar{t} = \bar{e}$ $\bar{\tau} \neq \bar{\varepsilon}$	T uses benign and malicious samples from different networks. E uses benign and malicious samples from different networks. T and E use malicious samples from the same network. T and E use the different attack classes.	Testing an 'extended' ML-NIDS devised by exploiting C7 against other attacks for which not enough samples are available to train a dedicated detector.
C9	$\bar{o} \neq \bar{t}$ $\bar{o} \neq \bar{e}$ $\bar{t} \neq \bar{e}$ $\bar{\tau} = \bar{\varepsilon}$	T uses benign and malicious samples from different networks. E uses benign and malicious samples from different networks. T and E use malicious samples from different networks. T and E use the same attack classes.	Using the benign samples of a different network and combining them with 'owned' malicious samples whether the 'owned' malicious samples
C10	$\bar{o} \neq \bar{t}$ $\bar{o} \neq \bar{e}$ $\bar{t} \neq \bar{e}$ $\bar{\tau} \neq \bar{\varepsilon}$	T uses benign and malicious samples from different networks. E uses benign and malicious samples from different networks. T and E use malicious samples from different networks. T and E use different attack classes.	Training a ML-NIDS using benign and malicious sample from two distinct subnets, and testing the generalization capabilities of such ML-NIDS against other attacks targeting a different subnet.

additional security risks. Indeed, mixing data from different networks presents several fundamental issues, which must be known when real ML-NIDS deployments are considered. We stress that our paper lies at the intersection of diverse research fields (i.e., network traffic analysis, machine learning, cybersecurity) and some of the following issues may be well-known within each field. Considered the scope of our paper, it is meaningful to make the entire community aware of such issues.

We identify the following three performance-related implementation *challenges*:

1) *Removing Network Artifacts:* Depending on the considered set of features, some samples may contain 'artifacts' that are unrelated to their benign/malicious nature.[5] If not sanitized, such artifacts may be learned by the ML model to perform its decisions, leading to overfitting and, hence, useless ML-NIDS.
2) *Preserving Performance:* When a given context involves modifications of *T*, it is important not to degrade the baseline False Positive Rate (FPR). Modifications of *T* must always be assessed.
3) *Maximizing Performance:* Assuming that simply adding malicious samples to *T* results in a ML-NIDS capable of detecting such attacks is misleading: it has been

[5]The most blatant example is when a dataset has all its malicious samples originating from the same IP address. If the IP address is considered as a feature, the ML model will only look for the 'malicious' IP address, meaning that any attack involving other machines will never be detected.

shown that ML models for NIDS may yield underwhelming detection performance in multi-classification settings [11]. It is hence crucial to consider a ML-NIDS architecture that optimizes the usage of such additional samples.

Finally, we highlight three intrinsic *risks* that involve security aspects of cross-evaluations of ML-NIDS.

- *Labeling quality:* Cross-evaluations are significant only if the samples in $\mathbb{N}$ and in $\mathbb{M}$ all report the correct label (i.e., benign or malicious). If, e.g., $\mathbb{N}$ contains malicious samples because the authors of the source dataset did not perform proper verifications, then the final results may be unreliable. Unless the cross-evaluation involves unsupervised ML, real deployments should ensure that all samples are associated to the correct ground truth.
- *Exposure to adversarial ML attacks:* Although mixing data from different networks can result in more resilient ML-NIDS (cf. Section II-B), relying on *public* datasets exposes to 'poisoning' attacks [51]. In these circumstances, training a ML-NIDS on such data would have the opposite effect of adversarial training. For instance, in [52] the FPR increases by 5 times when only 5% of the data is polluted. More subtle poisoning strategies exploit 'backdoors' which make ML-NIDS prone to evasion, as evidenced in [53]—and also in [54] for federated learning scenarios. Countermeasures include verifying the checksum of each dataset as provided by the authors; or applying some modifications that can remove or mitigate the effects of such poisoned samples (e.g., [55]).
- *Incompatible Networks:* Regardless of the resulting performance, mixing samples from different networks may not be possible a-priori. If the goal is using an N from a different network, it is necessary to conduct *preliminary* analyses ensuring that the two networks are indeed similar. On the other hand, when using M from different networks it is necessary to perform *follow-up* analyses that question the validity of the cross-evaluation results. This is because high detection rates at test-time may lead to a 'false sense' of security if the malicious activities depend on the underlying network's behavior. Such analyses may include comparing the feature importance between different models; or complete sanity checks by deploying the ML-NIDS against true attacks–in real time. Regardless, using the same source of benign samples both in T and E ensures that the FPR after deployment will not deviate from the one at test-time.

We make a crucial remark. Our cross-evaluation idea assumes that T and E always use benign data originating from the same network (i.e., o), which is in stark contrast with the practice of 'transferring ML models' (common in Computer Vision [56]). Indeed, we advise *against* using such practice for ML-NIDS, due to the immense variability of each network [9].

We can conclude that the additional context types enabled by cross-evaluations of ML-NIDS are intriguing, but practical applications are not simple and require the adoption of a rigorous workflow.

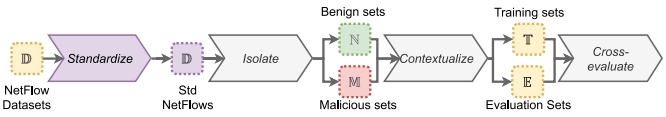


Fig. 3. Overview of XeNIDS. The input $\mathbb{D}$ is a set of labelled NetFlow datasets of n distinct networks. The output results should be further analyzed.

IV. Proposed Framework: XeNIDS

We showed that cross-evaluations of ML-NIDS are enticing but challenging, and we are not aware of efforts that tackled this problem in an exhaustive way. As a first step, we propose XeNIDS, a framework for the Cross-evaluation of Network Intrusion Detection Systems based on machine learning, with a focus on NetFlow data. Our proposed framework is rooted in the same design principles described in Section III-A, and has a threefold goal:

- allowing the *simulation* of all contexts in Table I;
- facilitating assessments of *multiple* contexts;
- addressing the challenges discussed in Section III-C.

Of course, we do not claim that XeNIDS is the only way to do all of the above. Our intention is to further promote the diffusion of cross-evaluations in *research*, as well as to increase their realistic value for proactive assessments.

We provide an overview of XeNIDS in Section IV-A, which consists in four stages: *standardize* (Section IV-B), *isolate* (Section IV-C), *contextualize* (Section IV-D), *cross-evaluate* (Section IV-E).

A. Overview

The focus of XeNIDS is on NetFlows (Section II-C), enabling *inter-compatibility* with PCAP data. NetFlows are metadata generated from packet captures, and summarize the communications between two endpoints. A NetFlow is defined as:

$$\text{NetFlow} = (srcIP, dstIP, srcPort, dstPort, t, proto, d, \dots,), \quad \text{(Exp. 3)}$$

where *srcIP* (*srcPort*) and *dstIP* (*dstPort*) are the source and destination IP addresses (network ports) of the two involved hosts, t is the timestamp of the first connection, d is the duration of the communication session, *proto* is the network protocol of the communication. Depending on the NetFlow software and its configuration, additional metrics can be computed: the most typical fields include the number of *packets* and *bytes* exchanged during the communication [57].

We present a schematic representation of XeNIDS in Fig. 3.

XeNIDS requires a set of n datasets containing NetFlows, representing $\mathbb{D}$ and totalling μ distinct attack classes.

These datasets must be provided with the *ground truth*. XeNIDS assumes that all data in $\mathbb{D}$ is verified, trusted and appropriate for the considered deployment scenario (Section III-C).

The framework includes four stages (cf. Fig. 3):

1) *Standardize:* the input datasets in $\mathbb{D}$ are first cleaned and sanitized, and then brought into a common 'language'.

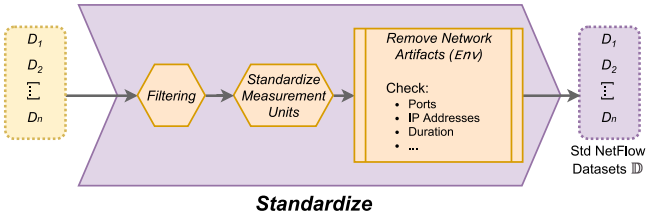


Fig. 4. First stage: Standardize. The initial NetFlows in $\mathbb{D}$ are all standardized to derive a common feature set, and cleaned of any possible artifact that may lead to overfitting and impractical ML-NIDS.

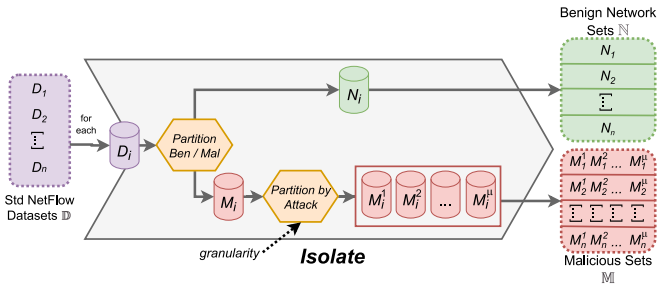


Fig. 5. Second stage: isolate. The standardized $\mathbb{D}$ is used to extract the n individual benign N and malicious M sets. The latter are then split in μ sets on the basis of their attack class. Depending on the user-provided granularity, it is possible to aggregate sets of different attack classes into a bigger class, therefore changing the initial μ. The outputs are the full sets $\mathbb{N}$ and $\mathbb{M}$.

2) *Isolate:* every standardized dataset is partitioned in its benign and malicious sets ($\mathbb{N}$ and $\mathbb{M}$).
3) *Contextualize:* $\mathbb{M}$ and $\mathbb{N}$ are used to compose a context (by generating the corresponding $\mathbb{T}$ and $\mathbb{V}$).
4) *Cross-evaluate:* $\mathbb{T}$ and $\mathbb{V}$ are used to develop and cross-evaluate a ML-NIDS.

The results provided as *output* by XeNIDS should be *further analyzed* for practical deployments (cf. Section III-C).

B. Standardize

In the first stage, schematically depicted in Fig. 4, XeNIDS brings all the datasets $D_i \in \mathbb{D}$ into a common NetFlow format, accounting for potential obfuscations as a result of *anonymization* techniques. Essential operations involve data sanitization (e.g., handling missing values) and filtering: for example, if the goal is the detection of attacks involving TCP traffic, then all non-TCP traffic can be safely removed. Then, the focus is on establishing a common feature set[6] while simultaneously *removing network artifacts* that may lead to overfitting. Such procedures are tough, especially when considering NetFlow records, but necessary. To explain the reasons of such difficulties and our proposed workarounds, we provide an *original interpretation* of NetFlows with respect to machine learning.

In simple terms, a NetFlow is the result of two contributors: the *communications* (*Comm*) performed by the involved hosts, and the effects of the *environment* (*Env*) where the NetFlow is generated. This latter factor (*Env*) is, in turn, influenced by two elements: the network *identity* (*NetId*), denoting the intrinsic characteristics of the network where *Comm* (such as allocated bandwidth, protocols used, common open ports, periodic services) are captured; and the *configuration* of the appliance (*Conf*) used to generate the NetFlows. Hence, the information captured by a NetFlow is a function[7] $\mathcal{F}$ of three components: *Comm*, *NetId*, *Conf*. Formally:

$$\mathcal{F}(Comm, Env) \Rightarrow \mathcal{F}(Comm, NetId, Conf) = \text{NetFlow} \quad \text{(Exp. 4)}$$

The ultimate goal of the standardize stage is to mitigate the effects of *Env* (represented by *NetId* and *Conf*) across all the input datasets in $\mathbb{D}$. Indeed, if one dataset D_i has an *Env* that is significantly stronger than D_j, then a ML model trained on data from D_i and D_j may only learn on the basis of such 'signature' *Env*. These circumstances lead to overfitting on *Env*, resulting in impractical detectors that neglect to search for malicious behaviors.

Let us explain Exp. 4 with two practical use-cases on the contribution of *Env*.

- Different *NetId*. Consider two different networks where a host downloads the same file from the same remote server via SSH: if these two networks use different listening ports for the SSH server (e.g., 22 and 4022), then the NetFlows of the first network will differ from those of the second network (they will have different ports).
- Different *Conf*. Using different NetFlow software and/or settings yields different NetFlows even when the original PCAP traces are identical. For instance, measurement units can differ, resulting in datasets that are not compatible: a dataset D_i with d expressed in milliseconds cannot be used alongside a dataset D_j that uses seconds.

We report in Appendix A an exhaustive explanation of the effects brought by *Env* on NetFlows.

By referring to the official NetFlow v9 documentation,[8] we observe that there are several fields that can contribute to *Env* (influenced both by *NetId* and *Conf*), which require particular care at this stage. We provide in Appendix B some recommendations for reducing the generation of the above-mentioned artifacts, with a focus on three fields: the IP address, the network ports, and the flow duration.

Nonetheless, depending on the considered use-cases, many low-level implementations are viable to minimize the impact of *Env* and derive a common feature set. After this stage, the initial set of datasets $\mathbb{D}$ is standardized and ready for the 'core' functionalities of XeNIDS.

C. Isolate

In this stage, XeNIDS isolates the benign from malicious samples of each (standardized) dataset in $\mathbb{D}$ to derive $\mathbb{N}$ and $\mathbb{M}$ (cf. Fig. 2 in Section III-A). We provide a schematic in Fig. 5.

Specifically, XeNIDS first partitions the benign from malicious samples in each D_i, resulting in two distinct sets, N_i and M_i. Then, XeNIDS further partitions the specific attack

[6]Taken from the *intersection* of the features across all $\mathbb{D}$.

[7]The definition of $\mathcal{F}$ is software dependent [57], and outside our scope.

[8]www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/

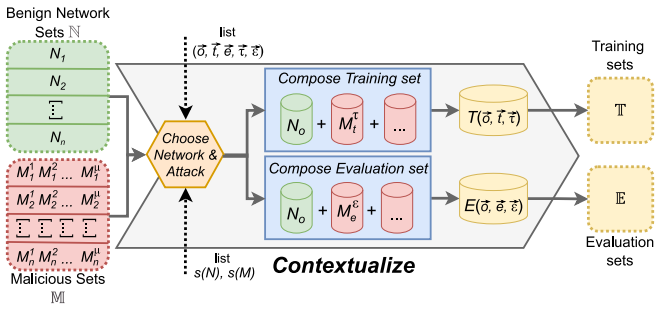


Fig. 6. Third stage: contextualize. XeNIDS uses $\mathbb{N}$ and $\mathbb{M}$ to create multiple T and E, according to the context-related parameters $(\vec{o}, \vec{t}, \vec{e}, \vec{\tau}, \vec{\varepsilon})$ and the splits $s(N)$ and $s(M)$. All the composed T and E are inserted in $\mathbb{T}$ and $\mathbb{E}$. **Example.** Assume that the following user-provided parameters: $\vec{o} = (1)$, $\vec{t} = (2)$, $\vec{e} = (3)$, $\vec{\tau} = (1)$, $\vec{\varepsilon} = (1)$; $s(N) = (80:20)$, $s(M) = (70:30)$. XeNIDS chooses N_1, and puts 80% of N_1 in T and the remaining 20% in V. XeNIDS then selects M_2^1 and puts 70% of its samples in T; then XeNIDS selects M_3^1 and puts 30% of its samples in V. Such T (and V) is then inserted in $\mathbb{T}$ (and $\mathbb{V}$). The operation is repeated if the user provides additional lists as input.

samples in M_i according to the *individual* attack that they represent[9] (assuming that $\mu > 1$).

Such design choice enables the development of *collaborative ensembles* (e.g., [19], [58]) of ML classifiers, each devoted to a specific threat, therefore addressing the third challenge (cf. Section III-C)—while also allowing to use XeNIDS for multi-classification ML problems.

We note that the separation can also account for a specific level of *granularity*. In this case, the original μ will be changed by 'aggregating' attacks of different classes, potentially even treating all malicious samples as belonging to a single malicious class. Depending on the use-case, such granularity can vary: it could either be performed at a high-level (e.g., *Botnet* or *DoS* attacks) or go at a deeper level (e.g., a specific *Botnet* variant). The final choice depends on the actual use-case (e.g., when there are not enough samples available, they can be aggregated into a macro-class, or simply discarded).

This stage produces two outputs: $\mathbb{N}$, containing all the benign network samples (partitioned in n subsets according to their source dataset); and $\mathbb{M}$, containing all malicious samples isolated in $n \times \mu$ subsets of samples according to their specific attack and source dataset. We recall that some elements of $\mathbb{M}$ can be empty, i.e., if a D_i does not contain malicious samples of the same classes as D_j.

D. Contextualize

In the third stage, XeNIDS creates *all* the sets corresponding to the contexts to simulate during the cross-evaluation. This is done by using $\mathbb{N}$ and $\mathbb{M}$ (provided by the previous stage), alongside some external input, to compose training T and evaluation E sets; all such T and E will be put in two dedicated collections, $\mathbb{T}$ and $\mathbb{E}$. We provide a schematic of this stage in Fig. 6.

Two user-provided input *lists* regulate this stage: a 5-dimensional tuple of context-related parameters $(\vec{o}, \vec{t}, \vec{e}, \vec{\tau}, \vec{\varepsilon})$; and a pair of *splits*, $s(N)$ and $s(M)$, used to partition (e.g., 80:20) any N and M that will be included in a given T and E. The idea is to facilitate cross-evaluations that consider multiple contexts, by composing all the necessary T and E *before* using them for any assessment. Hence, XeNIDS iterates over all the elements in the two input lists: at each iteration, XeNIDS composes a training and evaluation set according to the user-specified parameters.

Specifically, for each tuple $(\vec{o}, \vec{t}, \vec{e}, \vec{\tau}, \vec{\varepsilon})$, and for each pair of splits, $s(N)$ and $s(M)$, XeNIDS proceeds as follows.

- XeNIDS uses o to select a specific set of benign samples N_o from $\mathbb{N}$. XeNIDS splits N_o according to $s(N)$, and puts the corresponding partitions in T and E.
- For each $(t, \tau) \in (\vec{t}, \vec{\tau})$, XeNIDS extracts from $\mathbb{M}$ the element M_t^τ, which is split with $s(M)$ and put in T.
- For each $(e, \varepsilon) \in (\vec{e}, \vec{\varepsilon})$, XeNIDS extracts from $\mathbb{M}$ the element M_e^ε, which is split with $s(M)$ and put in E.
- Altogether, these operations result in two sets, $T(\vec{o},\vec{t},\vec{\tau})$ and $E(\vec{o},\vec{e},\vec{\varepsilon})$, which are put in $\mathbb{T}$ and $\mathbb{E}$.

An example of such workflow is in the caption of Fig. 6.

In cases where *t=e* and $\tau = \varepsilon$, XeNIDS performs the partitioning simultaneously, to avoid overlaps that can result in the same malicious samples being included in both T and E.

The selection of $s(N)$ and $s(M)$, which can differ for T and E, must be done to achieve a twofold goal: (i) realize an E that is comprehensive enough to cover the real data distribution, and hence produce insightful results; and (ii) realize a T that allows to develop proficient ML-NIDS. For instance, if E does not contain many benign samples, then the resulting FPR may not correspond to the real FPR after the ML-NIDS is deployed, At the same time, if T contains only a small number of samples for a given attack, the resulting ML-NIDS will not be able to capture all the possible variations of such attack.[10]

After this stage, we obtain two collections of training and evaluation sets, $\mathbb{T}$ and $\mathbb{E}$.

E. Cross-Evaluate

In the last stage, XeNIDS performs the cross-evaluation by using the sets in $\mathbb{T}$ and $\mathbb{E}$ to reproduce any user-specified context. Hence, the input parameters are a list of contexts C (cf. Exp. 2); as well as a learning ML algorithm to develop the detectors of the ML-NIDS.

Specifically, for each context $\mathsf{C}(\vec{o}, \vec{t}, \vec{e}, \vec{\tau}$ provided as input, XeNIDS draws the corresponding $T(\vec{o}, \vec{t}, \vec{\tau})$ and $E(\vec{o}, \vec{e}, \vec{\varepsilon})$ from $\mathbb{T}$ and $\mathbb{E}$. Then, depending on the architecture of the ML-NIDS, XeNIDS operates as follows.

- If the ML-NIDS leverages a *single classifier*, XeNIDS uses $T(\vec{o}, \vec{t}, \vec{\tau})$ to train a single (multi-class) ML-model with a given ML algorithm; such ML-model is then tested against $E(\vec{o}, \vec{e}, \vec{\varepsilon})$.
- If the ML-NIDS leverages *ensembles of classifiers*, XeNIDS splits $T(\vec{o}, \vec{t}, \vec{\tau})$ into smaller sets, (e.g., by composing $T(o, t, \tau)$ focusing on the specific attack τ contained in t); each of these sets is used to train a dedicated ML-model of the ensemble. Such procedure can be

[9]The attack is denoted by the ground-truth labels provided in the input $\mathbb{D}$.

[10]We refer the reader to [45] for a study on how the size of the training set can impact the performance of ML-NIDS.

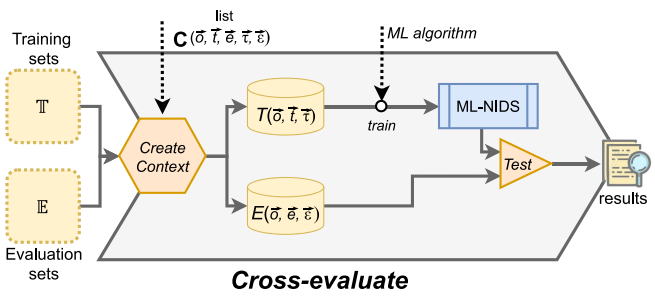


Fig. 7. Final stage: cross-evaluate. XeNIDS reproduces the user-specified context(s) and performs the cross-evaluation. **Example:** assume the following input context: C((1), (1,2), (1,2), (1,1), (2,2)) and a ML-NIDS that leverages ensembles of binary detectors. XeNIDS first extracts T((1),(1,2),(1,1)) from $\mathbb{T}$, which is split in two smaller sets, T(1,1,1) and T(1,2,1). Such sets are used to train two ML-models that will compose the ML-NIDS. The ML-NIDS can then be tested either against E((1),(1,2),(2,2)), or against its subsets E(1,1,2) and E(1,2,2), all of which obtained from $\mathbb{E}$. Any previously trained model (e.g., the one using T(1,1,1)) can be reused to assess different contexts.

repeated for $E(\vec{o}, \vec{e}, \vec{\varepsilon})$, i.e., the ML-NIDS can be tested against the entire E, or against subsets.

The design of XeNIDS enables the assessment of multiple contexts without the need of training additional ML models. If two contexts require the same T, it is only necessary to draw a different E from $\mathbb{E}$, and use such E to assess the previously trained ML-NIDS.

We illustrate this stage in Fig. 6, where we also provide a complete example of an ensemble use-case. We anticipate that, in our demonstration, we will always use ensembles of specialized classifiers.

The results produced as output of this stage should be subject to subsequent analyses and considerations.

V. Application

As a final contribution of this paper, we showcase[11] a practical application of XeNIDS. We do so via a large set of experiments where we cross-evaluate ML-NIDS by using a total of 6 well-known NID datasets. We describe our testbed (Section V-A) and explain the preprocessing operations (Section V-B). Then, we present the common assessment procedure (Section V-C).

A. Testbed

The aim of our demonstration is reproducing and assessing the use-cases described in Section II-B2. To this purpose, we assess three different context types (cf. Table I), namely C1, C4 and C7. However, we differentiate our experiments depending on the *format* of the NetFlow data used as input to XeNIDS: specifically, such data can be either in a *uniform* or *heterogeneous* format. Let us explain our rationale and the differences between these two distinct scenarios.

We recall that XeNIDS operates on *existing* data in the form of NetFlows. Such NetFlows can be provided either (a) as PCAP traces, and then exported to NetFlows using dedicated software; or (b) directly as NetFlows, processed according to the creators' specifications. These two scenarios must be treated separately, due to the different effects that they can have on the *results*. In the first scenario, the raw PCAP traces (collected in diverse network environments) can be used to generate *uniform* NetFlows by using the same appliance for all PCAP traces; because the NetFlows share the same format, the results are more reliable due to a lower chance of network artifacts (the contribution of *Conf* is the same—cf. Exp. 4). However, such scenario requires *all* source data to be fully provided as PCAP, which is a requirement that is hard to meet.[12] Therefore, it is insightful to consider also the scenario where the source datasets are provided directly as *heterogeneous* NetFlows (due to being generated with different software). Such scenario requires a more careful application of XeNIDS's *standardize* stage (Section IV-B), but also a more detailed analysis of the results because the effects of different initial *Conf* can only be seen *after* the ML-NIDS is evaluated.

We hence apply XeNIDS differently for both scenarios, each considering 4 well-known datasets ($n = 4$ for both scenarios).

- *Heterogeneous scenario:* here, we use the `CTU13` [16], `NB15` [60], `IDS18` [15], `DDOS19` [59]. These are all provided as NetFlows, but using different appliances.
- *Uniform scenario:* here, we use the `UF-BotIoT`, `UF-NB15`, `UF-IDS18`, `UF-ToNIoT`. These datasets are created in [21] by the PCAP version of existing datasets and generating the corresponding (labelled) NetFlows using a unified appliance.

For comparison purposes, two datasets are shared,[13] whereas two are unique for each scenario.

We provide in Table II an overview of these datasets. For each dataset, we report the amount of NetFlows, the overall number of malicious classes, the size of the provided feature-set, and the performance (as F1-score) achieved by the state-of-the-art. From this table, we can already observe the effects of *Conf* on the corresponding NetFlows: two datasets (i.e., the `NB15` and the `IDS18`) are used by both scenario, but the amount of samples and features differ. For example, the `NB15` has 2.5M samples in the *heterogeneous* scenario, and 1.6M in the *uniform* scenario. Moreover, let us focus on the performance achieved by past works. We can see that the state-of-the-art reaches very high F1-scores, which can raise the question of whether there is any point in improving such values. Nevertheless, we stress that cross-evaluations have a different objective (cf. Section II-B): assessing the effectiveness of ML-NIDS against different attacks (*not included* in the

TABLE II
Statistics of the Analyzed NID Datasets

Scenario	Dataset	#Samples	#Attacks	#Features	F1-score
Heter.	`CTU13`	20.7M	5	14	99.1% [17]
	`NB15`	2.5M	9	48	98.7% [18]
	`IDS18`	3.1M	14	80	96.2% [42]
	`DDOS19`	70M	18	80	99.0% [19]
Uniform	`UF-BotIoT`	600K	4	12	97.0% [21]
	`UF-NB15`	1.6M	9	12	85.0% [21]
	`UF-IDS18`	8.3M	14	12	83.0% [21]
	`UF-ToNIoT`	1.4M	9	12	100.0% [21]

[11] Our implementation of XeNIDS: https://github.com/pajola/XeNIDS.

[12] e.g., PCAP data can be truncated [16] or not fully labelled [59].

[13] `UF-NB15` and `UF-IDS18` are generated from `NB15` and `IDS18`.

TABLE III
DISTRIBUTION OF ATTACKS IN EACH DATASET. IN OUR IMPLEMENTATION OF XENIDS, WE *Always* USE THE SPECIFIC ATTACK CLASSES, AND PERFORM NO MERGING

Heterogeneous scenario				*Uniform* scenario			
Dataset	***Botnet***	***DoS***	***Other***	**Dataset**	***Botnet***	***DoS***	***Other***
CTU13	5	0	0	UF-BotIoT	0	2	2
NB15	2	1	6	UF-NB15	2	1	6
IDS18	1	5	8	UF-IDS18	1	5	8
DDOS19	0	18	0	UF-ToNIoT	0	2	7
Total	8	24	14	Total	3	10	23

respective datasets), and – if necessary – improving such ML-NIDS against these attacks. As our results will show, most of these ML-NIDS will perform poorly against different attacks, but can be strengthened; such achievements, however, could only be obtained by cross-evaluations.

Overall, these datasets contain traffic captured in large networks and the included malicious samples belong to a broad range of attacks[14] Table III shows the attack distribution of the input $\mathbb{D}$ for both scenarios. For simplicity, we organize Table III on the basis of three 'families' of attacks:

- *DoS*, for Denial of Service attacks (e.g., *DoS-Hulk*);
- *Botnet*, for Botnet attacks (e.g., *Rbot*);
- *Other*, for remaining attacks (e.g., *shellcode*, *scanning*).

We remark that, in our implementation of XeNIDS, we *always* use the specific attack classes, i.e., we do not 'aggregate' multiple attacks into a single class. The differentiation provided in Table III is for comprehensiveness, because the amount of specific attacks of our testbed is very broad. As an example, NB15 (and, hence, UF-NB15) has samples for all families, i.e., 2 different types of *Botnet* attacks, 1 type of *DoS*, and 6 types of *Other* attacks; whereas CTU13 only has samples for 5 different attacks of the *Botnet* family. From Table III we also determine that $\mu = 46$ in the *heterogeneous* scenario, and that $\mu = 36$ in the *uniform* scenario—this is because all the specific attack types are distinct across the input $\mathbb{D}$ datasets.

B. Preprocessing

We now describe the preprocessing computed on each considered NID dataset $D_i \in \mathbb{D}$ for both scenarios. Such operations represent the first two stages of XeNIDS: standardize (Section IV-B) and isolate (Section IV-C). Our low-level implementation of XeNIDS aims to overcome all the residual challenges in Section III-C—to the extent this is possible with the current state-of-the-art. The experimental platform is an Ubuntu 20.04 machine with 64GB RAM and an Intel Xeon E5-2620 CPU. The development leverages the Scikit-Learn suite.

Standardize: We first associate each sample to its ground truth.[15] Then, we derive a common feature set based on the official NetFlow v9 documentation, which we report in Table IV. These features represent the minimum set of common features obtainable from the source data for both scenarios. We note that some datasets are provided with more features (e.g., IDS18 has 80), which are left out. However, as we will show in our experiments, the considered features yield ML-NIDS with state-of-the-art performance.

TABLE IV
FEATURE SET OF OUR XENIDS IMPLEMENTATION IN BOTH SCENARIOS

#	Feature Name	Type
1	Source IP address internal	Bool
2	Destination IP address internal	Bool
3	Source port type	Cat
4	Destination port type	Cat
5	Flow Duration $[s]$	Num
6	Flow Direction	Bool
7	Incoming Bytes	Num
8	Outgoing Bytes	Num
9	Total Bytes	Num
10	Incoming Packets	Num
11	Outgoing Packets	Num
12	Total Packets	Num

To sanitize network artifacts, we follow the recommendations in Appendix B. To avoid overfitting and simulate the application of anonymisation techniques, we do not use the plain IP addresses or service-ports as features. Instead, we differentiate between internal/external hosts of each network (features 1 and 2 in Table IV); and we categorize the network ports according to the IANA guidelines (features 3 and 4 in Table IV). All of these operations are also adopted by recent works (e.g., [17]). We set the d of all samples in seconds, and we ensure that most samples fall within the same duration range (i.e., [0-150]s), discarding the few outliers.

Isolate: For each dataset D_i in $\mathbb{D}$, we separate benign from malicious samples using the ground truth label. We do not make any aggregation, hence our μ are the original ones (i.e., $\mu = 46$ for the *heterogeneous* scenario, and $\mu = 36$ for the *uniform* scenario). We thus obtain the following:

- for the *heterogeneous* scenario, $\mathbb{N}$ containing 4 elements representing the source networks of the respective datasets (CTU13, NB15, IDS18, DDOS19), and $\mathbb{M}$ containing 184 elements (because $n = 4$ and $\mu = 46$);
- for the *uniform* scenario, $\mathbb{N}$ containing 4 elements representing the source networks of the respective datasets (UF-BotIoT, UF-NB15, UF-IDS18, UF-TonIoT), and $\mathbb{M}$ containing 144 elements (because $n = 4$ and $\mu = 36$).

XeNIDS can now create the contexts to be cross-evaluated.

C. Assessment

In both scenarios we analyse three context types: C1, C4 and C7 (cf. Table I). Let us explain the common assessment procedures, focusing on the architecture of the ML-NIDS and the considered performance metrics.

Parameters and Performance Metrics: We use the same parameters for our implementation of XeNIDS. Specifically, the adopted splits $s(N)$ and $s(M)$ are always 80:20 for both T and E. We use such splits because they are common in related literature (e.g., [17], [45]), therefore enabling a more fair comparison of our results with those of past works. We considered different ML algorithms, but we found that Random Forests consistently provided the best tradeoff in

[14]For a precise description of each attack, we refer the reader to the source material provided by the creators of each dataset.

[15]We verify the checksum of each dataset, if provided.

terms of detection performance, rate of false alarms, and training time—a result that confirms the state-of-the-art on the same datasets (e.g., [17], [19], [21], [45]). Hence our results will refer to Random Forest as the learning algorithm for each classifier. The performance metrics of interest are the *F1-score* (F1) and the *False Positive Rate* (FPR), defined as follows:

$$\text{F1} = \frac{tp}{.5(fp+fn)+tp}, \qquad \text{FPR} = \frac{fp}{tp+fn}, \text{ (Exp. 5)}$$

where *tp*, *fp*, *fn* denote true positives, false positives, and false negatives, respectively; we consider a "true positive" as the correct detection of a malicious sample. It is desirable that the application of XeNIDS when considering modifications of the training set (hence, C7) should preserve the baseline FPR (cf. Section III-C). Finally, to account for the randomness of each split, we repeat each experiment 5 times, and in our results, we will report the average of each repetition.

ML-NIDS Architecture: XeNIDS fosters development of *ensembles* of detectors (Section IV-D), each specialized in a single attack. There are many ways in which such detectors can be integrated in a ML-NIDS. In our implementation, we assume that the NIDS uses ML as a final confirmation of detection. Hence, each sample is forwarded to the 'most suitable' detector of the ensemble, which must determine whether such sample is really malicious or not. While such selection is straightforward for contexts of type C1 and C7 (because $\bar{\tau} = \bar{\varepsilon}$), this is not the case for C4, where the ML-NIDS is tested against 'unknown' attacks (because $\bar{\tau} \neq \bar{\varepsilon}$). Hence, for C4 we perform a preliminary exploratory operation to identify the most suitable detector of the ensemble against the unknown attacks; this is to allow a more fair comparison with [19], which also investigates C4 by using two datasets considered in our testbed, `IDS18` and `DDOS19`. Hence, we reserve a portion of the samples of each unknown malicious class, and test every detector composing the ML-NIDS against such portion: the one with the best performance is chosen as the candidate for analyzing the corresponding attack. This is legitimate because the ground truth of such samples is known, and such samples (i) are never used in *E*, and (ii) are not added in *T* (otherwise it would not be C4). Therefore, when presenting the corresponding results, we will report the performance achieved by the most optimal detector against each specific attack.

VI. Demonstration

Our demonstration aims to simulate the exemplary use-cases described in Section II-B2. Let us discuss how we organize our demonstration by using the three considered types of contexts (C1, C4 and C7) enabled by the proposed model (cf. Table I).

Workflow: We follow the same workflow for both the *uniform* and *heterogeneous* scenario.

1) *Baseline (Section VI-A):* We begin by assessing the case where the organization $\mathcal{O}$ has an *N* and a *M* collected in their own network *o*. Such setup corresponds to context C1. To simulate C1, we use XeNIDS to devise a ML-NIDS for each dataset; such ML-NIDS is composed by an ensemble of detectors, each trained on a single attack contained in the same dataset. The ML-NIDS is tested against all the attacks of the 'origin' dataset. The expectation is that the results match the state-of-the-art.
2) *Generalization Section VI-B:* Having a ML-NIDS, the organization $\mathcal{O}$ wants to assess its effectiveness against different attacks not included in *T* and originating from a different network than *o*. Such setup corresponds to context C4. We use XeNIDS to *test* the 'baseline' detectors of C1 against *all* attacks of *all* datasets. The expectation is that the performance will decrease substantially.
3) *Extension (Section VI-C):* To compensate for the low performance against unknown attacks, the organization $\mathcal{O}$ borrows more malicious samples to improve the detection capabilities of their ML-NIDS. This corresponds to context C7 where $\bar{t}$ extends $\bar{o}$. For each dataset, XeNIDS trains additional detectors by using the malicious samples of *all* the other datasets, and adds such detectors to the ensemble of the 'baseline' ML-NIDS. Such 'extended' ML-NIDS is tested against *all* attacks of the CS. The expected result is an improved performance w.r.t. C4.
4) *Surrogation (Section VI-D):* If the organization $\mathcal{O}$ only has benign samples *N* from their own network *o* but does not have an *M*, the only option is using an *M* from a different network to develop a 'surrogate' ML-NIDS. This is also represented by C7, but in this case $\vec{o}$ and $\vec{t}$ are disjointed. Hence, for each dataset, XeNIDS uses *only* the additional detectors developed at the previous step to devise a (new) 'surrogate' ML-NIDS. Such surrogate ML-NIDS is then tested only against the 'attacks from different networks.

Example: Let us provide a complete example of the workflow above. We adopt the viewpoint of an organization that owns the `UF-NB15` network (hence, the *uniform* scenario). A total of 9 attacks originate from such network: 2 botnets, 1 DoS, and 6 others (cf. Table III).

1) XeNIDS uses the 9 attacks of `UF-NB15` to train 9 detectors, representing the baseline ML-NIDS, which is tested against these 9 attacks.
2) XeNIDS tests the baseline ML-NIDS (with its 9 detectors) against all the attacks of all datasets. Namely: 4 attacks for `UF-BotIoT`, 14 attacks for `UF-IDS18`, 9 attacks for `UF-ToNIoT`, as well as the 9 in `UF-NB15`.
3) XeNIDS trains 27 additional detectors, each using the benign samples of `UF-NB15` alongside the malicious samples of a specific attack contained in `UF-IDS18`, `UF-ToNIoT`, `UF-BotIoT`, respectively. Such detectors are *combined* with the 9 'baseline' detectors of `UF-NB15`, to extend the ML-NIDS. Such 'extended' ML-NIDS is tested against *all* the attacks of *all* datasets (36 attacks).
4) XeNIDS uses only the 27 detectors trained in the previous step (representing the 'surrogate' ML-NIDS) and tests them against the attacks contained in the corresponding networks, i.e., without taking into account the attacks (and the detectors) in `UF-NB15`.

Such workflow is followed 4 times for both scenarios, each time by considering a different dataset as 'origin'.

TABLE V
BASELINE–C1. THE TABLE SHOWS THE F1-SCORE OF THE ML-NIDS FOR EACH ORIGIN NETWORK. THE PERFORMANCE MATCHES THE STATE-OF-THE-ART. THE FPR IS LESS THAN 0.001 FOR ALL NETWORKS ASIDE FROM UF-BOTIOT (FPR = 0.11)

Heterogeneous scenario				*Uniform* scenario			
Dataset	***Botnet***	***DoS***	***Other***	**Dataset**	***Botnet***	***DoS***	***Other***
CTU13	98.1	—	—	UF-BotIoT	—	99.9	92.0
NB15	88.6	98.5	98.1	UF-NB15	83.4	91.4	95.8
IDS18	90.1	99.9	96.1	UF-IDS18	99.9	99.1	99.2
DDOS19	—	99.9	—	UF-ToNIoT	—	99.9	99.7

TABLE VI
GENERALIZATION–C4. EACH BASELINE ML-NIDS OF C1 IS TESTED AGAINST THE ATTACKS OF ALL OTHER NETWORKS. MOST ATTACKS ARE NOT DETECTED, AND THE F1-SCORE DEGRADES. THE FPR IS THE SAME AS IN C1 BECAUSE THE BENIGN SAMPLES ARE ALWAYS THE SAME AND THE TRAINING SET IS NOT MODIFIED

Heterogeneous scenario				*Uniform* scenario			
Dataset	***Botnet***	***DoS***	***Other***	**Dataset**	***Botnet***	***DoS***	***Other***
CTU13	80.0	38.1	49.7	UF-BotIoT	47.8	69.0	76.8
NB15	65.8	40.7	75.2	UF-NB15	72.2	52.3	64.1
IDS18	54.9	49.4	76.1	UF-IDS18	68.2	81.0	63.3
DDOS19	54.4	99.5	83.1	UF-ToNIoT	82.1	89.3	85.1

A. Baseline

We start by assessing C1, and report the results in Table V. Specifically, on the left, we present the results for the *heterogeneous* scenario, and on the right, the *uniform* scenario. For each dataset, we report the average F1-score obtained against each family of attacks (cf. Table III). Moreover, we report in the captions the average *FPR* achieved by the ML-NIDS of each dataset. Henceforth, all our results will be reported in the same format as Table V.

From Table V, we observe that there are only two scores for UF-BoTIoT and UF-ToNIoT, because there are no *Botnet* samples in these 'origin' datasets. Similarly, CTU13 and DDOS19 presents only one score.

All our baseline detectors match the performance of past works (cf. Table V). As an example, for the *uniform* scenario, the 'worst' ML-NIDS is trained (and evaluated) on UF-NB15, but also in [21] such ML-NIDS achieves an average F1-score of 85%. Similarly, in the *heterogeneous* scenario, the ML-NIDS in [19] achieve 99.0 F1-score on both IDS18 and DDOS19, whereas [17] achieves 99.0 F1 on CTU13—all these results align with ours, confirming that our XeNIDS implementation is efficient.

B. Generalization

We then assess the baseline ML-NIDS when they are subject to attacks also contained in different networks, i.e., C4. We report the detection results in Table VI; the FPR is the same as in the baseline C1 (cf. caption of Table V): this is expected because in C4 uses the same training sets as C1, and also the benign samples of the evaluation sets are the same as in C1,

From Table VI, we observe that the performance decreases because most of the attacks are 'unknown' to the baseline ML-NIDS. However, we can observe some interesting phenomena.

In the *uniform* scenario, the ML-NIDS of UF-ToNIoT can detect botnet attacks remarkably well (82% F1-score), despite *having no* ML-model specialized on botnet attacks (because no such attacks are contained in UF-ToNIoT). Such an intriguing finding could only be appreciated by cross-evaluating the ML-NIDS trained on UF-ToNIoT against malicious samples from different networks. Furthermore, the *heterogeneous* scenario shows that the baseline ML-NIDS of DDOS19 works very well against DoS attacks from other networks—despite such attacks being performed by different means.

We also compare some of our results with those in [19], which also investigated C4. Specifically, the ML-NIDS trained on DDOS19 and tested on IDS18 in [19] achieves 64% F1-score on average, which is similar to ours. Conversely, the ML-NIDS trained on IDS18 and tested on DDOS19 in [19] achieves an average 78% F1-score, which is slightly superior than ours. We explain this difference to the different conditions in [19]: they only consider a smaller portion of the initial dataset, whereas we use all of them. Hence, our samples may present a more skewed distribution that makes them more difficult to classify.

TABLE VII
EXTENSION–C7. BY AUGMENTING THE TRAINING SET OF THE ML-NIDS WITH THE MALICIOUS SAMPLES, THE F1-SCORE IMPROVES W.R.T. C4. THE AVERAGE FPR IS LOWER THAN 0.001 FOR ALL NETWORKS ASIDE FROM UF-BOTIOT (FPR = 0.01)

Heterogeneous scenario				*Uniform* scenario			
Dataset	***Botnet***	***DoS***	***Other***	**Dataset**	***Botnet***	***DoS***	***Other***
CTU13	98.8	99.9	98.9	UF-BotIoT	99.7	99.9	99.2
NB15	97.1	99.9	99.1	UF-NB15	88.9	99.2	98.7
IDS18	98.5	99.7	97.7	UF-IDS18	99.9	99.4	97.8
DDOS19	99.9	99.9	98.6	UF-ToNIoT	99.7	99.9	99.9

C. Extension

Next, we assess C7 when $\bar{\tau}$ extends $\bar{o}$, and report the results in Table VII. We observe that the overall performance increases (w.r.t. Table VI) by augmenting the training sets with the corresponding malicious samples.

In the heterogeneous scenario, our 'extended' ML-NIDS naturally outperform those in [19], but we cannot claim this as a contribution because our 'extended' ML-NIDS use an augmented training set.

We also appreciate that the FPR remains stable (cf. Table V). We owe such results to our reliance on ensembles of detectors.

D. Surrogation

Finally, we assess C7 when $\bar{\tau}$ and $\bar{o}$ are disjointed, and report the results in Table VIII. From this table, we observe that all detectors exhibit very high F1-scores, implying that the malicious samples are considerably different than the benign samples. Sometimes, the F1-score reaches 99.9%, but is not perfect: we believe such occurrence to be *positive* because an F1-score of 100% could be related to overfitting.

VII. DISCUSSION

We now discuss the results presented in Section VI. We first summarize the main findings (Section VII-A), and then make some considerations *reliability* of the results (Section VII-B

TABLE VIII
SURROGATION–C7. WE EXCLUDE ALL MALICIOUS SAMPLES (AND DETECTORS) FROM THE EACH 'ORIGIN' NETWORK. THE EXTREMELY HIGH PERFORMANCE MUST BE INVESTIGATED. THE AVERAGE FPR IS LESS THAN 0.001 AND 0.0001 FOR ALL NETWORKS OF *Uniform* AND *Heterogeneous* SCENARIOS, RESPECTIVELY

Heterogeneous scenario				*Uniform* scenario			
Dataset	***Botnet***	***DoS***	***Other***	**Dataset**	***Botnet***	***DoS***	***Other***
CTU13	99.9	99.9	98.9	UF-BotIoT	99.7	99.9	99.9
NB15	99.9	99.9	99.9	UF-NB15	99.9	99.9	99.9
IDS18	99.6	99.6	99.8	UF-IDS18	99.9	99.9	99.9
DDOS19	99.9	99.9	98.6	UF-ToNIoT	99.7	99.9	99.9

and Section VII-C). We then present the main limitations of our demonstration, as well as possible workarounds (Section VII-D).

A. Preliminary Analysis

We appreciate that, in general, our results show the effectiveness of XeNIDS in producing baselines with state-of-the-art performance, while also extending the detection surface. It is intriguing that, in some cases, it is possible to detect attacks without training on the related malicious samples. To further stress the advantages of cross-evaluations, we provide an in-depth look at our results by focusing on the CTU13 dataset. This dataset contains *only botnet* attacks and, from Table II, the state-of-the-art (e.g., [17]) achieves 99.1% F1-score against such attacks. A similar performance may suggest that improvements can be incremental at best; however, no past works have assessed how ML-NIDS trained on CTU13 can detect different *botnet* attacks (not included in CTU13). By applying the proposed XeNIDS framework, we discover that similar ML-NIDS perform much worse: as shown by Table VI (Section VI-B), the F1-score of such ML-NIDS drops by 20% against *botnet* attacks of diverse datasets; even worse, it is unable to detect *DoS* attacks (F1-score of 38%). *Such poor performance could only be assessed via cross-evaluations.* To make it better, the performance against these – different – attacks can be increased by training on the respective samples: by observing Table VII, the F1-score can be restored to 99% via cross-evaluations. Also noteworthy is that the FPR always remains within acceptable levels (below 0.001). Such FPR will resemble the one after deployment (because the source of benign samples is always the same).

However, as stated in Section III-C, it is necessary to further analyze the results of XeNIDS. This is to avoid relying on a false-sense of security, given by high performance at test-time which does not correspond to the performance after the ML-NIDS is deployed. We specifically focus on contexts of type C7 because they involve modifications of the training data, which can lead to 'network artifacts' that affects the *Env* component of NetFlows (cf. Exp. 4 in Section IV-B) and, potentially, lead to overfitted ML-NIDS.

B. Reliability: Uniform Scenario

In this scenario, by definition, the *Env* is affected only by *NetId* because *Conf* is the same for all datasets; such characteristic implicitly reduces the risk of network artifacts.

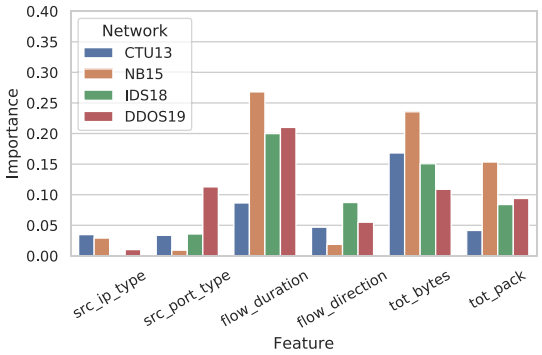


Fig. 8. Feature importances of the *Rbot* detectors (Heterogeneous scenario).

Nevertheless, we find instructive to analyze the results of the surrogate ML-NIDS, reported in the right-side of Table VIII. In particular, we consider the UF-NB15 network. We observe that the 'surrogate' detectors focused on botnet attacks achieve a near-perfect F1-score, which is *higher* than both their 'extended' and 'baseline' variants (cf. Tables VII and V). This implies that benign samples of UF-NB15 are very similar to the (malicious) botnet samples of UF-NB15, making such botnet samples harder to classify by the UF-NB15 ML-NIDS w.r.t. the botnet samples in other networks. Such occurrence can be a sign of overfitting, because the UF-NB15 ML-NIDS could be detecting the botnet samples from other networks on the basis of network artifacts. However, a more detailed analysis can remove such doubt. Indeed, in the uniform scenario, the only other source of 'botnet' samples is UF-IDS18, where the baseline performance is also perfect (cf. Table V), a result also confirmed by the state-of-the-art [21]. Simply put, the 'botnet' samples in UF-IDS18 are easy to identify. Such observation reduces the chance that the surrogate (or the extended) ML-NIDS of UF-NB15 are affected by artifacts from UF-IDS18.

C. Reliability: Heterogeneous Scenario

This scenario assumes NetFlows generated via different means, hence the *Env* component is affected by both *NetId* and *Conf*. Such characteristic increases the chance that some artifacts 'evaded' XeNIDS standardize stage. To find a trace of such artifacts, we compare the *feature importances* of each ML-NIDS (*all* ML-NIDS use the same feature set).

Intuitively, the most important features for detecting an attack in its 'origin' network should denote the malicious behavior–hence, such features should be also the most important when the attack is 'transferred' to train a different ML-NIDS (which is the case in C7). We provide in Fig. 8 a comparison of such importances, focusing on the detectors specialized on the *Rbot* botnet attack (contained in the CTU13 network). Specifically, Fig. 8 shows the importances of the top6 most important features (out of 12–cf. Table IV) for all the *Rbot* detectors among the four different networks.

From Fig. 8 we observe that the detectors can either 'agree' or 'disagree' on the importance of such features. Specifically, we observe that the 'origin' CTU13 detector (blue bars)

places a great importance on the *tot_bytes*, denoting agreement with the other detectors; however, there is disagreement on the *duration*, which is less important for the CTU13 detector. The general trend in Fig. 8 is that the detectors disagree on most features: therefore, we cannot exclude that some underlying effects of *Env* are still present.

D. Limitations and Future Work

To increase the reliability of the detection performance, it is necessary to assume the perspective of the *owners* of each network. As a practical example that could remove any doubt, the owners of the NB15 network should infect their machines with the *Rbot* botnet (contained in CTU13), and verify whether their ML-NIDS (trained on the *Rbot* samples from CTU13) can detect such attack. Doing such verifications is not possible for our scientific paper, as they require a complete control and overview of the monitored network. Moreover, the CnC servers of the *Rbot* botnet are no longer active. Our experiments are for demonstrative purposes, but realistic deployments should integrate such verifications–which *must* be done regardless of the origin of the malicious samples (i.e., both in 'traditional' and in 'cross' evaluations).

Moreover, we note that each considered context type is an independent use case. Indeed, our focus is not on developing systems that outperform the state-of-the-art: it would be unfair to claim that our ML-NIDS generated via C7 are better than those in [19]. In contrast, our goal is to demonstrate the contexts that can be assessed by mixing different network datasets, showcasing the potential of such cross-evaluation for the state-of-the-art. As such, we can consider our results as a 'benchmark', allowing future cross-evaluation studies to compare their results with those in our paper.

Finally, an intriguing future research direction is the assessment of cross-evaluations in adversarial settings: for instance, how would a ML-NIDS poisoned with samples from a different network perform (cf. Section III-C)? Answering a similar question would be beneficial to the ML-NIDS research area.

VIII. Conclusion

Despite many successes, the integration of supervised Machine Learning (ML) methods in Network Intrusion Detection Systems (NIDS) is still at an early stage. This is due to the difficulty in obtaining comprehensive sets of labelled data for training and evaluating a ML-NIDS. The recent release of labelled datasets for ML-NIDS was appreciated by the research community; however, few works noticed the opportunity that such availability provides to the state-of-the-art.

Inspired by the necessity of proactive empirical evaluations and the recent release of more open datasets, we promote the idea of cross-evaluating ML-NIDS by using existing labelled data from different networks. Such approach has been applied before, but no past work specifically tackled this problem. As a result, all the benefits of cross-evaluations, as well as their intrinsic risks, are still unexplored.

We address all of these issues in this paper. We begin by presenting the first model for cross-evaluation of ML-NIDS, which is data-agnostic and general enough to cover both supervised and unsupervised ML-NIDS. By using such model, we highlight the limited scope adopted by most related works, and showcase the benefits provided by cross-evaluations of ML-NIDS. We also present all the challenges and limitations of such opportunity, which must be known and adequately addressed in order to provide actionable results.

To foster proactive cross-evaluations, we develop XeNIDS, the first framework for cross-evaluations of ML-NIDS. XeNIDS aims to mitigate all the hazards arising from using data from different networks. Specifically, XeNIDS focuses on NetFlow data, which is popular in the ML-NIDS community due to its flexibility and suitability for detection purposes.

Finally, we elucidate the potential of cross-evaluations via a large set of experiments, where we use XeNIDS to cross-evaluate ML-NIDS on 6 well-known datasets. In our demonstration, we show the capability of XeNIDS to retain the 'baseline' performance of past ML-NIDS, while illustrating some additional use-cases enabled by cross-evaluations, such as 'extending' the detection surface of ML-NIDS. We conclude our demonstration with a follow-up discussion where we question the reliability of the results, which is necessary for realistic deployments of ML-NIDS.

Our paper will hopefully inspire future works on ML-NIDS, and is oriented to both researchers and practitioners. The former can make better use of open datasets to cross-evaluate past and future ML-NIDS, allowing broader assessments of the state-of-the-art; the latter can use future research results, or completely integrate cross-evaluations in their proactive assessments, to develop or improve Machine Learning-based Network Intrusion Detection Systems—without incurring in extra labeling procedures. We believe that cross-evaluations – supported by data-sharing platforms and federated learning techniques – represent a pragmatic way to overcome the specificity of NIDS and realize 'general' ML-NIDS.

Appendix A
Contributors to NetFlows

Let us illustrate the role played by *Comm* and *Env* (i.e., *NetId* and *Conf*) in the generation of the corresponding NetFlows (see Exp. 4). Assume that two organizations, $\mathcal{O}_1$ and $\mathcal{O}_2$, have two distinct networks both having a pair of hosts (h_1^1 and h_1^2 for $\mathcal{O}_1$, h_2^1 and h_2^2 for $\mathcal{O}_2$); such hosts communicate with each other within their own networks. It is straightforward that if these two pairs of hosts exchange different information (viz., resulting in different *Comm*) then the resulting NetFlows generated in $\mathcal{O}_1$ and $\mathcal{O}_2$ will differ. Let us focus on the case where the pairs of hosts exchange the same information (viz., same *Comm*). For simplicity, assume that the first host of each pair (h^1) sends *exactly* the same file of 100MB to the second host, using *exactly* the same protocol and ports. Let us assume that the hosts in $\mathcal{O}_1$ are allocated a bandwidth of b_1 Mb/s, and that those in $\mathcal{O}_2$ are allocated a bandwidth of b_2 Mb/s. Finally, let us assume that the organizations use the same NetFlow generation software, configured to allow the maximum duration of a NetFlow to be d_1 for $\mathcal{O}_1$, and d_2 for $\mathcal{O}_2$. We identify four scenarios.

- $b_1 = b_2$ and $d_1 = d_2 \rightarrow$ same *NetId* and same *Conf* (viz. same *Env*). For instance, if $b_1 = b_2 = 100$Mb/s and $d_1 = d_2 = 10$s, then the file will be transferred in the same amount of time (8s) in both $\mathcal{O}_1$ and $\mathcal{O}_2$, resulting in similar NetFlows (with a duration of 8s).
- $b_1 \neq b_2$ and $d_1 = d_2 \rightarrow$ different *NetId* but same *Conf* (viz. different *Env*). For instance, if $b_1 = 100$Mb/s and $b_2 = 1$Mb/s, then the file will be transferred in 8s in $\mathcal{O}_1$ but in 800s in $\mathcal{O}_2$, resulting in different NetFlows.
- $b_1 = b_2$ and $d_1 \neq d_2 \rightarrow$ same *NetId* but different *Conf* (viz. different *Env*). For instance, if $b_1 = b_2 = 100$Mb/s while $d_1 = 10$s and $d_2 = 1$s, then the transfer will take 8s in both $\mathcal{O}_1$ and $\mathcal{O}_2$; but in $\mathcal{O}_1$ there will be 1 NetFlow of 8s, while in $\mathcal{O}_2$ there will be 8 NetFlows of 1s.
- $b_1 \neq b_2$ and $d_1 \neq d_2 \rightarrow$ different *NetId* and different *Conf* (viz. different *Env*). This is self-explanatory.

Of course, there are many other factors that affect *NetId* and *Conf* (aside from the bandwidth and maximum duration). The above-mentioned example is just for demonstrative purposes.

APPENDIX B
GUIDELINES FOR STANDARDIZE

To avoid generating network specific artifacts (cf. Section IV-B), we provide some recommendations on three common NetFlow fields: the *IP addresses*, the *service ports*, and the *duration*.

IP addresses: There are two issues that may arise when standardizing the IP addresses of two distinct datasets:

- different networks use different subnet masks. For instance, the internal IP addresses of D_i may present the structure "192.168.x.x", whereas those in D_j are "175.32.x.x";
- the malicious traffic of a given dataset may be entirely produced by just few machines.

Neglecting these issues may result in ML models that distinguish legitimate from anomalous samples on the sole basis of the IP address of a host, without giving the due importance to the remaining traffic characteristic. This is a problem because if a *real* attack involves a machine with a different IP address, the detector would never identify it. We hence propose to standardize each dataset by separating *internal* from *external* hosts. The ML model will use these features, instead of the IP addresses, to perform its analyses. The information to perform this separation can be obtained either from the documentation of a dataset, or by inferring it from the data using expert knowledge; if such information is not obtainable, then we suggest not to use any IP-related feature.

Service Ports: Handling the service ports of distinct datasets presents similar issues to the IP addresses discussed above: different networks may adopt different port policies; and the attacks captured by a given dataset may rely just on a restricted (or unique) set of ports. We thus propose to standardize each dataset by categorizing each port on the basis of the IANA guidelines, i.e., *well-known* [0-1023], *registered* [1024-49151] and *dynamic* [49151-65535].

Duration: Besides verifying that all datasets use the same measurement units, standardizing the NetFlow duration (d) of distinct datasets is a challenging task. On the one hand, datasets may be created with different NetFlow tools and/or different configuration parameters. For example, setting the maximum duration (d_{max}) of a NetFlow to 1000 or 100 seconds would lead to significantly different results.[16] On the other hand, there may be some underlying traits of a given network that lead its machines to generate flows of different duration. To address these issues, we propose three possible solutions, all involving the identification of the smallest maximum duration across all datasets, $min(d_{max})$:

- *Outlier removal*. This approach assumes that (i) the duration of the majority of samples (from all considered datasets) falls within a reduced range $[d_l, d_t]$, and that (ii) the top limit d_t of this range is *lower* than $min(d_{max})$. In these circumstances, it is possible to remove the few "outliers" that have extremely high durations with respect to the remaining samples. Despite the consequential loss of samples, removing outliers does not necessarily reduce the prediction performance.
- *Threshold setting*. This solution avoids data loss problems. If a dataset D_i has $d_{max} \gg min(d_{max})$, its samples having $d > min(d_{max})$ will have their duration set to $min(d_{max})$. However, it is important to store the original value of d if it is needed to compute some derived metrics, such as the *packets per second*. This approach may be unpractical for ML leveraging sequential analyses as it disrupts the sequence of samples.
- *Flow splitting*. This technique enables the application of sequential ML methods. The intuition is to *split* those flows that exceed $min(d_{max})$. Given a D_i with $d_{max} > min(d_{max})$, the idea is to truncate all flows of D_i with duration $d > min(d_{max})$ into multiple flows. As a practical example assuming duration expressed in seconds, if $min(d_{max}) = 300$ and D_i has $d_{max} = 1000$, and if a given flow in D_i has $d = 700$, the approach truncates this flow in three distinct flows, with $d = (300, 300, 100)$. When performing the split, it is important to also update some metrics, such as the transferred bytes or packets (which can be adjusted proportionally) as well as the start and finishing times of the flow.

We observe that, in our experiments, we adopt the *outlier removal* strategy. Despite being lossy, such technique still allows to devise ML-NIDS with performance matching the state-of-the-art (see Table V and compare it with Table II).

APPENDIX C
SYMBOL TABLE

To facilitate the readability, we report in Table IX the major notation used throughout the main sections of our paper.

We also further explain the difference between some of our symbols introduced in Section III, and specifically the difference between the arrays and sets (e.g., $\vec{t}$ and $\bar{t}$). Let us assume a scenario where $n = 3$ and $\mu = 3$, meaning that $\mathbb{M}$ is a 3×3 matrix. We use the *ordered arrays* $\vec{t}$, $\vec{\tau}$ (or $\vec{e}$, $\vec{\varepsilon}$) to answer the question "which elements of $\mathbb{M}$ are included in T (or E)?".

[16] This issue can be overcome if the datasets are provided in PCAP format by properly setting the NetFlow generation tool.

TABLE IX
TABLE OF RELEVANT NOTATION USED IN THIS PAPER

SYMBOL	DESCRIPTION	REF
$\mathcal{O}$	An *organization*	§II-B2
o	The *network* of the organization $\mathcal{O}$	§II-B2
N	A set of *benign* network samples	§II-B2
M	A set of *malicious* network samples	§II-B2
$\mathbb{D}$	A *collection* of datasets, each generated in a specific network	§III-A
n	The *cardinality* of $\mathbb{D}$, i.e., the number of different networks included in $\mathbb{D}$	§III-A
D_i	The set of samples generated by network i ($\bigcup_i D_i = \mathbb{D}$)	§III-A
N_i, M_i	The set of benign and malicious samples included in D_i ($N_i \cup M_i = D_i$)	§III-A
μ	The number of all attacks included in $\mathbb{D}$	§III-A
M_i^α	The samples of network i corresponding to the attack α ($\bigcup_\alpha M_i^\alpha = M_i$)	§III-A
$\mathbb{N}$, $\mathbb{M}$	The collection of all N and M included in $\mathbb{D}$	§III-A
T, E	The *training* and *evaluation* sets of a ML-NIDS	§III-B
N_o	The benign samples (from the same network o) used in both T and E	§III-B
M_i^τ	An element of $\mathbb{M}$ used in T	§III-B
M_e^ε	An element of $\mathbb{M}$ used in E	§III-B
$\bar{t}$, $\bar{e}$	The set of all networks included in T and E	§III-B
$\bar{\tau}$, $\bar{\varepsilon}$	The set of all attacks included in T and E	§III-B
$T(\bar{o}, \bar{t}, \bar{\tau})$	The function describing the training set T	§III-B
$E(\bar{o}, \bar{e}, \bar{\varepsilon})$	The function describing the evaluation set E	§III-B
$\mathtt{C}(\bar{o}, \bar{t}, \bar{e}, \bar{\tau}, \bar{\varepsilon})$	A *context* is defined by the relationships between $\bar{o}$, $\bar{t}$, $\bar{\tau}$, $\bar{e}$, $\bar{\varepsilon}$	§III-B
$Comm$	The contribution to a NetFlow of the *communications* of the involved hosts	§IV-B
Env	The contribution to a NetFlow of the network *environment* of its two hosts	§IV-B
$NetId$	The *intrinsic properties* of a network influencing Env	§IV-B
$Conf$	The *configuration* of the NetFlow appliance influencing Env	§IV-B
$\mathbb{T}$, $\mathbb{E}$	The set of all T and E generated by XeNIDS	§IV-D

A possibility is that $\vec{t} = (1, 1, 2)$ and that $\vec{\tau} = (2, 3, 3)$. This means that T will contain M_1^2, M_1^3, M_2^3. Hence, $\bar{t} = (1, 2)$ because $\bar{t}$ is the *set* denoting the (unique) 'malicious' networks included in T. At the same time, $\bar{\tau} = (2, 3)$ because $\bar{\tau}$ is the *set* denoting the (unique) attacks included in T.

Finally, we stress that, in our cross-evaluation model, $\bar{o} = \vec{o} = o$, because the origin of the benign samples must be the same for both the training and evaluation partitions (i.e., T and E, respectively).

REFERENCES

[1] J. A. M. Sidey-Gibbons and C. J. Sidey-Gibbons, "Machine learning in medicine: A practical introduction," *BMC Med. Res. Methodol.*, vol. 19, no. 1, pp. 1–18, 2019.

[2] C.-J. Wu *et al.*, "Machine learning at Facebook: Understanding inference at the edge," in *Proc. IEEE Int. Symp. High Perf. Comput. Archit.*, Washington, DC, USA, 2019, pp. 331–344.

[3] K. Bresniker, A. Gavrilovska, J. Holt, D. Milojicic, and T. Tran, "Grand challenge: Applying artificial intelligence and machine learning to cybersecurity," *Computer*, vol. 52, no. 12, pp. 45–52, Dec. 2019.

[4] W. Fleshman, E. Raff, R. Zak, M. McLean, and C. Nicholas, "Static malware detection & subterfuge: Quantifying the robustness of machine learning and current anti-virus," in *Proc. 13th IEEE Int. Conf. Malicious Unwanted Softw.*, Nantucket, MA, USA, 2018, pp. 1–10.

[5] G. D'Angelo, M. Ficco, and F. Palmieri, "Malware detection in mobile environments based on autoencoders and API-images," *Elsevier J. Parallel Distrib. Comput.*, vol. 137, pp. 26–33, Mar. 2020.

[6] B. Liang, M. Su, W. You, W. Shi, and G. Yang, "Cracking classifiers for evasion: A case study on the Google's phishing pages filter," in *Proc. 25th Int. Conf. World Wide Web*, 2016, pp. 345–356.

[7] "Machine learning in the age of cyber AI," Darktrace, Cambridge, U.K., Rep., 2020. [Online]. Available: https://www.darktrace.com/es/resources/wp-machine-learning.pdf

[8] "Using AI to detect and contain cyberthreats," Lastline, Redwood City, CA, USA, White Paper, 2019. [Online]. Available: https://www.lastline.com/wp-content/uploads/2020/01/Lastline_WP_AI_Done_Right_web.pdf

[9] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proc. IEEE Symp. Security Privacy*, Oakland, CA, USA, 2010, pp. 305–316.

[10] B. Biggio, G. Fumera, and F. Roli, "Security evaluation of pattern classifiers under attack," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 4, pp. 984–996, Apr. 2014.

[11] G. Apruzzese, M. Colajanni, L. Ferretti, A. Guido, and M. Marchetti, "On the effectiveness of machine and deep learning for cyber security," in *Proc. IEEE 10th Int. Conf. Cyber Conflicts*, Tallinn, Estonia, May 2018, pp. 371–390.

[12] B. Miller *et al.*, "Reviewer integration and performance measurement for malware detection," in *Proc. Int. Conf. DIMVA*, 2016, pp. 122–141.

[13] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," *Comput. Security*, vol. 86, pp. 147–167, Sep. 2019.

[14] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli, "A detailed investigation and analysis of using machine learning techniques for intrusion detection," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 686–728, 1st Quart., 2019.

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

[16] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Elsevier Comput. Security*, vol. 45, pp. 100–123, Sep. 2014.

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

[18] M. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, "Multi-stage optimized machine learning framework for network intrusion detection," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, pp. 1803–1816, Jun. 2021.

[19] C. F. T. Pontes, M. M. C. de Souza, J. J. C. Gondim, M. Bishop, and M. A. Marotta, "A new method for flow-based network intrusion detection using the inverse potts model," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, pp. 1125–1136, Jun. 2021.

[20] C. Schaffer, "Selecting a classification method by cross-validation," *Mach. Learn.*, vol. 13, no. 1, pp. 135–143, 1993.

[21] M. Sarhan, S. Layeghy, N. Moustafa, and M. Portmann, "NetFlow datasets for machine learning-based network intrusion detection systems," in *Proc. EAI Int. Conf. Big Data Technol.*, 2021, pp. 117–135.

[22] L. Williams, G. McGraw, and S. Migues, "Engineering security vulnerability prevention, detection, and response," *IEEE Softw.*, vol. 35, no. 5, pp. 76–80, Sep./Oct. 2018.

[23] F. Pierazzi, G. Apruzzese, M. Colajanni, A. Guido, and M. Marchetti, "Scalable architecture for online prioritisation of cyber threats," in *Proc. IEEE 9th Int. Conf. Cyber Conflicts*, Tallinn, Estonia, May 2017, pp. 1–18.

[24] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1153–1176, 2nd Quart., 2016.

[25] R. J. Joyce, E. Raff, and C. Nicholas, "A framework for cluster and classifier evaluation in the absence of reference labels," in *Proc. 14th ACM Workshop Artif. Intel. Security (AI Security)*, 2021, pp. 73–84.

[26] E. Bursztein, M. Martin, and J. Mitchell, "Text-based CAPTCHA strengths and weaknesses," in *Proc. ACM Conf. Comput. Commun. Security*, 2011, pp. 125–138.

[27] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, "ImageNet training in minutes," in *Proc. Int. Conf. Parallel Process.*, 2018, pp. 1–10.

[28] E. Min, J. Long, Q. Liu, J. Cui, Z. Cai, and J. Ma, "SU-IDS: A semi-supervised and unsupervised framework for network intrusion detection," in *Proc. Springer Int. Conf. Cloud Comput. Security*, 2018, pp. 322–334.

[29] R. Jordaney *et al.*, "Transcend: Detecting concept drift in malware classification models," in *Proc. USENIX Security Symp.*, 2017, pp. 625–642.

[30] Z. Ahmad, A. S. Khan, C. W. Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Trans. Emerg. Telecommun. Technol.*, vol. 32, no. 1, p. e4150, 2021.

[31] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martinez-del Rincón, and D. Siracusa, "LUCID: A practical, lightweight deep learning solution for DDoS attack detection," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 2, pp. 876–889, Jun. 2020.

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

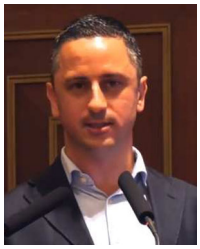
[33] C. Zhang, X. Costa-Pérez, and P. Patras, "Tiki-taka: Attacking and defending deep learning-based intrusion detection systems," in *Proc. ACM Conf. Cloud Comput. Security Workshop*, 2020, pp. 27–39.

[34] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Elsevier Pattern Recognit.*, vol. 84, pp. 317–331, Dec. 2018.

[35] M. Horák, V. Stupka, and M. Husák, "GDPR compliance in cybersecurity software: A case study of DPIA in information sharing platform," in *Proc. ACM Int. Conf. Availab. Reliab. Security*, 2019, pp. 1–8.
[36] P. Spagnolettia and A. Salvia, "Digital systems in high-reliability organizations: Balancing mindfulness and mindlessness," in *Proc. Int. Workshop Socio-Techn. Perspective Inf. Syst. Develop.*, 2020, pp. 155–161.
[37] R. Ramaswamy and T. Wolf, "High-speed prefix-preserving IP address anonymization for passive measurement systems," *IEEE/ACM Trans. Netw.*, vol. 15, no. 1, pp. 26–39, Feb. 2007.
[38] I. Dayan *et al.*, "Federated learning for predicting clinical outcomes in patients with COVID-19," *Nat. Med.*, vol. 27, pp. 1735–1743, Sep. 2021.
[39] F. Falcão *et al.*, "Quantitative comparison of unsupervised anomaly detection algorithms for intrusion detection," in *Proc. ACM Symp. Appl. Comput.*, 2019, pp. 318–327.
[40] M. Catillo, A. D. Vecchio, L. Ocone, A. Pecchia, and U. Villano, "USB-IDS-1: A public multilayer dataset of labeled network flows for IDS evaluation," in *Proc. IEEE Int. Conf. Depend. Syst. Netw.*, Taipei, Taiwan, 2021, pp. 1–6.
[41] A. Bansal and S. Mahapatra, "A comparative analysis of machine learning techniques for botnet detection," in *Proc. 10th Int. Conf. Security Inf. Netw.*, 2017, pp. 91–98.
[42] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41525–41550, 2019.
[43] A. Divekar, M. Parekh, V. Savla, R. Mishra, and M. Shirole, "Benchmarking datasets for anomaly-based network intrusion detection: KDD CUP 99 alternatives," in *Proc. IEEE 3rd Int. Conf. Comput. Commun. Security*, Kathmandu, Nepal, 2018, pp. 1–8.
[44] R. Magán-Carrión, D. Urda, I. Díaz-Cano, and B. Dorronsoro, "Towards a reliable comparison and evaluation of network intrusion detection systems based on machine learning approaches," *Appl. Sci.*, vol. 10, no. 5, p. 1775, 2020.
[45] Y. Zhang, J. Niu, G. He, L. Zhu, and D. Guo, "Network intrusion detection based on active semi-supervised learning," in *Proc. IEEE/IFIP Int. Conf. Depend. Syst. Netw. Workshops*, Taipei, Taiwan, 2021, pp. 129–135.
[46] C. G. Cordero, E. Vasilomanolakis, A. Wainakh, M. Mühlhäuser, and S. Nadjm-Tehrani, "On generating network traffic datasets with synthetic attacks for intrusion detection," *ACM Trans. Privacy Security*, vol. 24, no. 2, pp. 1–39, 2021.
[47] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: Detecting botnet command and control servers through large-scale NetFlow analysis," in *Proc. ACM Annu. Conf. Comput. Security Appl.*, Dec. 2012, pp. 129–138.
[48] J. Hou, P. Fu, Z. Cao, and A. Xu, "Machine learning based DDoS detection through NetFlow analysis," in *Proc. IEEE Mil. Commun. Conf.*, Los Angeles, CA, USA, 2018, pp. 1–6.
[49] G. Apruzzese, M. Marchetti, M. Colajanni, G. G. Zoccoli, and A. Guido, "Identifying malicious hosts involved in periodic communications," in *Proc. IEEE Int. Symp. Netw. Comput. Appl.*, Cambridge, MA, USA, Oct. 2017, pp. 1–8.
[50] M. Stevanovic and J. M. Pedersen, "An analysis of network traffic classification for botnet detection," in *Proc. IEEE Int. Conf. Cyber Situational Awareness Data Anal. Assessment*, London, U.K., Jun. 2015, pp. 1–8.
[51] G. Apruzzese, M. Andreolini, L. Ferretti, M. Marchetti, and M. Colajanni, "Modeling realistic adversarial attacks against network intrusion detection systems," *ACM Digit. Threats Res. Pract.*, to be published.
[52] C. Dunn, N. Moustafa, and B. Turnbull, "Robustness evaluations of sustainable machine learning models against data poisoning attacks in the Internet of Things," *Sustainability*, vol. 12, no. 16, p. 6434, 2020.
[53] M. Bachl, A. Hartl, J. Fabini, and T. Zseby, "Walling up backdoors in intrusion detection systems," in *Proc. ACM Workshop Big Data Mach. Learn. Artif. Intell. Data Commun. Netw.*, 2019, pp. 8–13.
[54] T. D. Nguyen, P. Rieger, M. Miettinen, and A.-R. Sadeghi, "Poisoning attacks on federated learning-based IoT intrusion detection system," in *Proc. Workshop Decentralized IoT Syst. Security*, 2020, pp. 1–7.
[55] G. Apruzzese, M. Colajanni, L. Ferretti, and M. Marchetti, "Addressing adversarial attacks against security systems based on machine learning," in *Proc. IEEE Int. Conf. Cyber Conflicts*, Tallinn, Estonia, May 2019, pp. 1–18.
[56] A. Oliver, A. Odena, C. A. Raffel, E. D. Cubuk, and I. J. Goodfellow, "Realistic evaluation of deep semi-supervised learning algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 3235–3246.
[57] G. Vormayr, J. Fabini, and T. Zseby, "Why are my flows different? a tutorial on flow exporters," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 2064–2103, 3rd Quart., 2020.
[58] T. Acharya, I. Khatri, A. Annamalai, and M. F. Chouikha, "Efficacy of heterogeneous ensemble assisted machine learning model for binary and multi-class network intrusion detection," in *Proc. IEEE Int. Conf. Autom. Control Intell. Syst.*, Shah Alam, Malaysia, 2021, pp. 408–413.
[59] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy," in *Proc. IEEE Int. Conf. Security Technol.*, Chennai, India, 2019, pp. 1–8.
[60] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proc. Mil. Commun. Inf. Syst. Conf.*, Canberra, ACT, Australia, 2015, pp. 1–6.

Giovanni Apruzzese received the master's *(summa cum laude)* and Ph.D. degrees in computer engineering from the University of Modena, Italy, in 2016 and 2020, respectively. He has been a Postdoctoral Researcher with the Institute of Information Systems, University of Liechtenstein since 2020. In 2019, he spent 6 months as a Visiting Researcher with Dartmouth College, Hanover, NH, USA, under the supervision of Prof. V. S. Subrahmanian. His research interests involve all aspects of big data security analytics with a focus on machine learning, and his main expertise lies in the analysis of network intrusions, phishing, and adversarial attacks. Homepage: https://www.uni.li/giovanni.apruzzese.

Luca Pajola received the M.Sc. degree in computer science from the University of Padova, Italy, in 2018. He is currently pursuing the Ph.D. degree with the School of Brain Mind and Computer Science, University of Padova, Italy. Here, he is part of the SPRITZ Security and Privacy Research Group research group under the supervision of Prof. M. Conti. He is conducting research on fields, including security and machine learning. Homepage: https://www.math.unipd.it/∼pajola/.



Mauro Conti (Fellow, IEEE) received the Ph.D. degree from the Sapienza University of Rome, Italy, in 2009. After his Ph.D., he was a Postdoctoral Researcher with Vrije Universiteit Amsterdam, The Netherlands. He is a Full Professor with the University of Padua, Italy. He is also affiliated with TU Delft and University of Washington, Seattle. In 2011, he joined as Assistant Professor with the University of Padua, where he became Associate Professor in 2015 and a Full Professor in 2018. He has been a Visiting Researcher with GMU, UCLA, UCI, TU Darmstadt, UF, and FIU. His research is also funded by companies, including Cisco, Intel, and Huawei. His main research interest is in the area of Security and Privacy. In this area, he published more than 400 papers in topmost international peer-reviewed journals and conferences. He has been awarded with a Marie Curie Fellowship (2012) by the European Commission, and with a Fellowship by the German DAAD (2013). He is the Editor-in-Chief of IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, the Area Editor-in-Chief for IEEE COMMUNICATIONS SURVEYS AND TUTORIALS, and has been an Associate Editor for several journals, including IEEE COMMUNICATIONS SURVEYS AND TUTORIALS, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, and IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT. He was Program the Chair of TRUST 2015, ICISS 2016, WiSec 2017, ACNS 2020, and CANS 2021 and the General Chair of SecureComm 2012, SACMAT 2013, NSS 2021, and ACNS 2022. He is the Senior Member of the ACM and a Fellow of the Young Academy of Europe. Homepage: https://www.math.unipd.it/∼conti/.