A Fast and Accurate Threat Detection and Prevention Architecture using Stream Processing

Antonio G. P. Lobato*, Martin Andreoni Lopez*, Alvaro A. Cardenas‡, Otto Carlos M. B. Duarte*, and Guy Pujolle†
*Universidade Federal do Rio de Janeiro - GTA/COPPE/UFRJ - Rio de Janeiro, Brazil
†Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6 - Paris, France
‡University of California at Santa Cruz, USA

***Abstract*—Late detection of security breaches increases the risk of irreparable damages and limit any mitigation attempts. We propose a fast and accurate Threat Detection and Prevention Architecture that combines the advantages of real-time streaming with batch processing over a historical database. We create a dataset by capturing both legitimate and malicious traffic and propose two ways of combining packets into flows, one considering a time window and the other analyzing the first few packets of each flow per period. We also investigate the effectiveness of our proposal on real-world network traces obtained from a significant Brazilian network operator providing broadband Internet to their customers. We implement and evaluate three classification algorithms and two anomaly detection methods. The results show an accuracy higher than 95% and an excellent trade-off between attack detection and false-positive rates. We further propose an improved scheme based on Software Defined Networks, that automatically prevents threats by analyzing only the first few packets of a flow. The proposal promptly and efficiently blocks threats, is robust, and can scale up, even when the attacker employs spoofed IP.**

I. Introduction

Communication networks transfer an increasing amount data due to billions of users and devices, creating several challenges for threat detection [1], [2], [3]. Moreover, Denial of Service (DoS) attacks already reached more than 1.35 terabits per second [4] and the longest in 2016 lasted 197 hours [5], [6]. Network providers have to handle millions of data streams in real-time, and attacks can attempt to hide in this deluge of information; therefore, detecting network attacks becomes a *needle in a haystack* problem. Besides, the time it takes to detect an attack is essential, and if detection takes too long, irreparable damages will occur. Current security systems are not effective, since 85% of network intrusions are detected weeks after they had happened [7], with an average detection time of 206 days. To address this problem, in this paper we leverage recently-developed "big data" frameworks for real-time *stream processing*, and show their effectiveness for detecting network threats by providing *fast* analysis of *large* and *diverse* network data.

In particular, we propose and implement a fast and accurate Threat Detection and Prevention Architecture that combines several big data open-source platforms for batch and stream processing. We propose a combination of (1) conventional *batch* processing over a historical database, with (2) real-time *stream* processing analysis to detect threats in real-time. We analyze the performance of three popular open-source stream processing platforms to choose the most effective one to detect threats as fast as possible.

For stream processing, we study two ways of combining the packets into flows, henceforward defined as a sequence of packets from the same source IP to the same destination IP within a time window. The first approach collects all packets in a time window, and the second approach uses only the first packets of each flow, *i.e.*. We periodically analyze the first few packets of each flow. Moreover, we propose a scheme based on Software Defined Networks (SDN) and the analysis of the first few packets of a flow to implement our Threat Prevention Architecture, to block threats with spoofed IP addresses. Also, the off-path analysis machines only analyze the first packets of each flow per period, being immune to flooding attacks.

To evaluate our proposal, we create a dataset with labeled classes for the architecture evaluation, containing normal network usage and several attacks. Besides, we use another dataset with real-world broadband user data from one of the most significant network operators in Brazil. We implement five detection methods, three supervised classification methods to detect known threats, and two unsupervised anomaly detection methods to detect zero-day attacks and unknown threats. The results show a high accuracy for known threats, higher than 95%, and an excellent trade-off between false positives and threat detection in the anomaly methods.

In summary, the proposed Threat Detection and Prevention Architecture as a whole presents several advantages, such as i) effective threat detection for both known and unknown threats; ii) fast counter-measures against threats, since it relies on stream processing and the rapid analysis of the first packets of a flow, without having to wait to the end of the flow; iii) robust against flooding attacks and with a high potential for scalability since the analysis machine only receives few packets per flow; and iv) effective in the threat network traffic block, since, with the use of Software Defined Networks, the architecture blocks the malicious traffic as near of its source as possible, even on scenarios with spoofed source IP addresses.

We organize the rest of the paper as follows. Section II discusses related work. Our proposed stream processing algorithms are presented in Section III. Both datasets to evaluate the architecture are presented in Section IV. In Section V, we discuss how to select the various parameters of our architecture and show the results of the different threat detection methods.

Section VI presents a resilient SDN-enabled architecture that can detect attacks accurately while surviving attempts by the attacker to obfuscate its activities and which also tries to attack our detection tools. Finally, Section VII concludes the work.

II. Related Work

There are different aproaches to detect denial of service attacks. One of them is to accomplished statiscal measures and trace back to the origins of the attack. The trace back procedure, however, requires to store information in the routers or to include information in every packet, which are complex operations [8], [9]. Our work is related to previous efforts on using *machine learning* and *big data* frameworks for detecting intrusions, and on software-defined networks to dynamically respond to detected threats.

Machine learning techniques can be either supervised or unsupervised, depending on whether the dataset is labeled or not. Algorithms that use supervised learning include neural networks, decision trees, and Support Vector Machines (SVM) [10]. These types of algorithms have been increasingly applied to computer security applications; for example, Li *et al.* combine a pattern matching technique, Dynamic Time Warping (DTW), and SVM to generate intrusion detection rules [11]. The method is evaluated with the traditional KDD dataset to classify between Denial of Service (DoS), Probe, and Remote to Local (R2L) attacks. Supervised learning, however, is trained with known attacks, and these algorithms sometimes have a hard time identifying previously unseen attacks. In unsupervised learning, the training data does not have labels, and the algorithms generally learn a representation of the patterns of the data. These algorithms are generally useful for anomaly detection; for example, Lakhina *et al.* propose the use of sample entropy for anomaly detection, and they show that this metric combined for source and destination IPs and ports, together with volume analysis can detect multiple sources of anomalies [12]. Common methods to detect outliers apply Principal Component Analysis (PCA) and Independent Component Analysis (ICA) when data is assumed to follow a non-gaussian distribution. Fernandes *et al.* use PCA combined with Ant Colony Optimization for clustering, to perform profile-based for anomaly detection [13]. Palmieri *et al.* use ICA to separate the network traffic from their different sources [14]. Then, decision trees are applied to perform binary classification. However, PCA and ICA are both sensitive to noise when used in anomaly detection [15]. Amaral *et al.* proposed an improved approach using Tsallis entropy to detect anomalous traffic [16]. The authors this technique to real and simulated traffic, but it only considers six features for anomaly detection. The authors use a graph representation of network features, allowing deep inspection of IP flows. In contrast with our parameter adaptation solution, this system needs parameters defined by the network administrator for its execution.

Chellammal and Malarchevi proposed an architecture that uses conventional algorithms in parallel and a feature selection mechanism to reduce datasize [17]. Nevertheless, the number of base learners used in the training phase and if they use the same or different base learns are application dependent, which makes complex the proposal deployment.

Villar-Rodriguez *et al.* propose the use of Support Vector Machine (SVM) to detect identity theft in social networks [18]. The authors monitor user profiles based on connection time information. SVM classifies the legitimate user and the attackers' profiles. Li *et al.* propose an approach to anomaly detection in traffic monitoring. The authors use Principal Component Analysis (PCA) over the Random Forest machine-learning algorithm to identify the most important features [19]. In our proposal, we improve the sample entropy metric employing a time series that takes into account 26

In addition to machine learning, the use of *big data* frameworks has also become popular in security applications. Bostani and Sheikan propose an unsupervised anomaly detection algorithm using MapReduce [20]. The work uses the Optimum-Path Forest (OPF) algorithm to project clustering models and detect anomalous behaviors. The paper only focuses on two specific Internet of Things (IoT) attacks, sinkhole and selective-forwarding, disregarding unknown threats or zero-day attacks. Singh *et. al.* proposed a similar platform in which the use of big data analytics leveraging Hadoop[1], to detect Peer-to-Peer botnets [21]. The offline analysis execution is the main limitation of this approach. BigFlow [22] is another proposal that employs stream processing for monitoring high-speed networks. The authors employ several classifiers to determine if a networking event is suspicious. Then, stream learning allows incremental model updates with new knowledge for the system. Even if the proposal can analyze events in a high-throughput, once a threat is detected, there is no protection system. In a previous work [23], we implement an intrusion detection system using stream processing and machine learning. Nevertheless, in this paper, we focus on providing real-time stream analytic to improve the time to detect attacks. Another related topic when using machine learning for security is how to evaluate the performance of these methods. Cardenas *et al.* discuss the tradeoff between precision and false alarms and propose intrusion detection operating characteristic (IDOC) curves to evaluate intrusion detection problems [24]. A classical dataset used for evaluating proposals is the KDD99 dataset [25].

A number of other works also create their dataset to evaluate their proposals [26], [27]. Sangkatsanee *et al.* [26] classify attacks using different algorithms, while Morteza Amini *et al.* [27] detect anomaly in real-time with unsupervised neural networks. These works, however, do not evaluate their proposals in scenarios with a large amount of traffic. On the contrary, we use two real traffic datasets (one with a large amount of traffic) with packet captures from our lab and one of the major network operators in Brazil.

The programmability of Software Defined Networking is also receiving more attention in the security community. We proposed BroFlow [28], [29], a combination of the Bro IDS and the OpenFlow POX controller. We implemented an algorithm to block a Denial of Service (DoS) attack by

[1]Hadoop is the Apache foundation framework to store and process data using MapReduce processing model.

using Bro to send messages to the controller, to mitigate the attack automatically. Nevertheless, the implemented countermeasure blocks all traffic generated by the identified source IP. Therefore, the system is ineffective under IP address spoofing. Lin *et al.* extend the SDN architecture for traffic classification and Intrusion Prevention [30]. The proposal detects DoS attacks and malicious HTTP requests. Nevertheless, the work mainly uses SDN functionalities for load balancing, and it does address automatic attack detection. OPCloudSec is another proposal that integrates SDN and cloud computing for security management [31]. The authors use a deep learning approach based on Deep Belief Networks for DDoS detection. The authors do not discuss the requirement of parameter tuning and disregard the effect of IP Spoofing when detecting DDoS. IP spoofing could easily overload routing tables in Software Defined Networks. Vincentini *et al.* use a similar approach to our proposal combining Apache Storm for streaming processing with Floodlight controller for SDN [32]. In contrast to our work focused on network security, the authors use this combination for network management in a multi-tenant environment.

In a companion conference paper, we present preliminary results of an adaptive threat detection architecture that uses a honeypot and trains detection models in real-time [33]. Our proposal extends our previous work in several aspects: i) we leverage big data stream processing frameworks, ii) we propose fast machine-learning algorithms for the first few packets of each flow, and, iii) we use software defined network to respond to an attack promptly. Unlike the previously cited papers, we propose a specific architecture that uses the lambda architecture, allowing real-time stream processing analysis based on the support of a historical database. Our architecture prevents threats in a fast and scalable way by providing real-time, accurate detection of known and zero-day attacks through automated classification and anomaly detection methods.

III. Proposed Threat Detection Architecture

Conventional *big data analytics* tools usually employ batch processing; however, batch processing produces high latency, with responses in the order of tens of seconds, which is unacceptable for critical applications requiring real-time processing, with fast responses within a second [34]. Unlike batch processing, stream processing techniques can analyze continuously generated data and provide real-time results. Their accuracy, however, might not be as good as batch methods that use more data. Both of these paradigms, batch, and stream, can be combined to exploit the benefits of each technique in the lambda architecture, which analyzes big data in a real-time and accurate manner [35].

Our proposed architecture, shown in Figure 1, is based on the lambda architecture because it combines both *batch* and *stream* processing methods. The lambda architecture has three layers: the stream processing layer, the batch-processing layer, and the service layer. The stream processing layer deals with the incoming data in real-time. The batch-processing layer analyzes a huge amount of stored data in a distributed way through techniques such as map-reduce. Finally, the service layer combines the obtained information of the two previous layers to provide an output composed of analytic data to the user. Therefore, the lambda architecture gives us the ability to analyze streaming data to obtain real-time results while at the same time complementing this analysis with historical data.

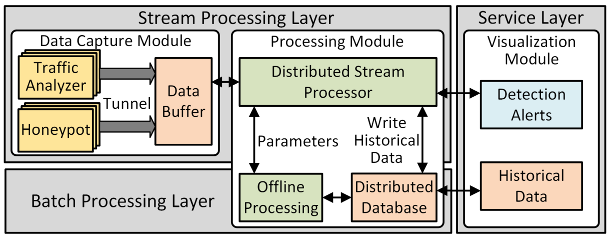


Figure 1: Proposed architecture for real-time stream processing and historical data batch processing composed of the following modules: i) the Data Capture Module gathers data, ii) the Processing Module analyzes and stores it, and the Visualization Module displays analytic information results.

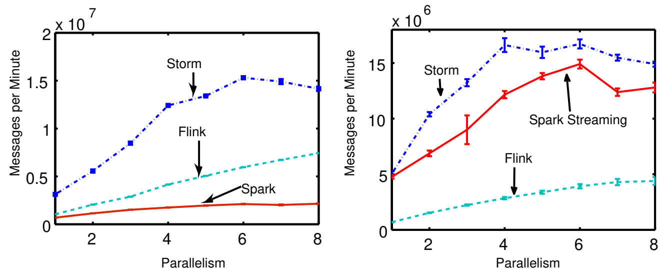
We implement the traffic analyzer using `Bro`[2], a real-time network security monitor with its network-oriented programming language that allows us to create programs to make flow abstraction easier. In our proposed architecture, `Bro` identifies network flows, extracts information from the packets, and groups them in time windows.

Since network usage varies over time, captured data arrives at different rates. Hence, an overload may occur while processing the streams, resulting in data loss. Apache `Kafka` receives the captured data, acting as a buffer to prevent overload at the analysis location. `Kafka` is a message broker that works as a publish/subscribe service and acts basically as a buffer to the processing tool, adapting different generation and processing rates. `Kafka` abstracts the message flow into topics. Producers then write their data into topics from which the consumers can access the information.

To analyze streaming data, we chose `Storm` over two other streaming platforms based on a performance evaluation we develop in Section III-A. `Storm` offers a distributed fault-tolerant stream processing framework. In addition, `Storm` processes data in memory, ensuring low latency in real-time. `Storm` processes the streams in a Direct Acyclical Graph (DAG) composed of input elements, called *Spouts*, and processing elements, called *Bolts*. The application can define the parallelism of the *Spouts* and *Bolts* in a way that multiple stream samples can be processed simultaneously.

Once we obtain analytics from the streams, we store the results in a dynamic database for historical analysis. We use a distributed database to achieve better resilience. Our architecture uses Apache's `HBase`, a fault-tolerant database that can store large quantities of sparse data. We calculated offline the parameters with historical data to improve the accuracy and adaptiveness of our threat detection. Then, we calibrate the processing model for real-time threat detection. The architecture has an adaptive characteristic, because the parameters are periodically updated, adapting to new network use patterns.

[2]The Bro project was renamed to Zeek project: https://www.zeek.org/



(a) Evaluation of the threat detection performance
(b) Evaluation of the wordcount performance.

Figure 2: Throughput results of the platforms in terms of number of messages processed per minute in function of the task parallelism.

A. Stream Processing Performance Evaluation

We evaluate three open-source streaming platforms with our threat detection application to choose the one that can process data at higher rates because one of our main concerns is to detect threats as fast as possible. We compare `Storm` against `Spark Streaming` and `Flink` [36]. We use our threat detection application as a benchmark to measure stream-processor performance. The attack-detection methods in our application are further presented in Section V. The experiment evaluates the performance of the platforms in terms of processing throughput. The dataset is injected into the platforms in its totality and replicated as many times as necessary. We measure the consumption of messages and the processing rate of each platform. We also vary the parallelism parameter, which represents the total number of cores available for the cluster to process samples in parallel. Figure 2a shows the performance results of this experiment. `Storm` shows a higher throughput when compared to the alternatives. Figure 2a also shows that `Storm` process up to 15 million samples per minute with our Threat Detection application, which gives about four microseconds of detection time, allowing defense strategies and significantly decreasing the risks.

In addition to this experiment, we also use another benchmark that counts the number of times each word appears in a text, using a dataset that contains more than 5,000,000 tweets [37]. All three platforms offer the word-count application as an example. Therefore, we show this result to get an unbiased comparison that is not affected by our implementation. Figure 2b shows the performance behavior of the three systems under a word-count program. Once again, `Storm` has a better performance and, therefore, is the adequate platform for our Threat Detection Architecture.

IV. Security Datasets

There are few openly-available network security datasets to evaluate defense mechanisms due to the privacy and security concerns with sharing real-world data. DARPA [38] is the first effort for creating open network intrusion data and its follow-up KDD 99 [25] dataset. The DARPA dataset includes TCP/IP traffic and operating system data collected from a simulated computer network. While collecting the data, attacks were also simulated and labeled in the dataset. The KDD 99 dataset consists of a selection and grouping of DARPA features to facilitate the application of machine learning algorithms. These datasets, however, consider a network simulation that is a limitation [39] because it introduces artifices in the training and testing dataset. Moreover, these datasets are two decades old and do not represent current threats [40].

In this work, we evaluate our threat detection proposal using two different datasets, both containing real-world traffic. One dataset contains traffic from a Brazilian Internet Service Provider, and the other contains real traffic from our lab. The use of two different datasets shows that the proposed architecture and its detection methods work well, even considering distinct scenarios. Moreover, we propose two modes of combining the packets into flows, both of them considering a flow as a sequence of packets from the same IP source to the same IP destination during a period. In the former, we gather all packets in a fixed-length time window. We determine the length of this window further in the paper, based on the accuracy of the classification algorithms. Each flow has 26 features, generated by the TCP/IP header data. The main features are TCP, UDP, and ICMP packet rates; source and destination port number; the number of each TCP flag; average and variance of inter-packet arriving time; average and variance of flow packet length; among others. The second mode to define flows consists of extracting the features from the first few packets of each flow. The intuition behind this approach is the well defined initial behavior of most applications, which leads to proper classification. Once again, we determine the number of packets to be considered based on the accuracy of the classification methods.

A. Network Operator Dataset

Real-world information from 373 residential broadband users from the city of Rio de Janeiro for a period of one week composes the Network Operator (NetOp) dataset. We anonymize the network traffic for privacy considerations. An Intrusion Detection System (IDS) filtered the traffic. We analyzed the logs from this IDS, and the proportion of traffic filtered out was around 15%. Since we obtained the data filtered by an IDS, we added real botnet malicious traffic captured in the work of García *et. al* [41] to detect these threats and evaluate our threat detection architecture. The botnet data has 13 different scenarios of malware infection. These attacks are real and were not performed by the authors since they infected the machines with real malware. In the combined dataset, we keep 15% threat traffic proportion.

B. GTA/UFRJ Dataset

Another contribution of this work is the creation of a dataset with real network traffic to evaluate the proposal[3]. The dataset has around 95 GByte of packet capture raw data in computers from our lab, GTA at Federal University of Rio de Janeiro. We added to the normal traffic, real network threats, including seven types of DoS and nine types of network

[3]The dataset can be obtained by emailing the authors.

probes. The analysis of packet header information detects two threat classes: Denial of Service (DoS) attacks and probe. Therefore, we elaborate on the dataset with several attacks from both these classes. The DoS attacks are *ICMP flood*, *land*, *nestea*, *smurf*, *SYN flood*, *teardrop*, and *UDP flood*. The different types of probes in the dataset are *TCP SYN scan*, *TCP connect scan*, *SCTP INIT scan*, *Null scan*, *FIN scan*, *Xmas scan*, *TCP ACK scan*, *TCP Window scan*, and *TCP Maimon scan*. We launch the attacks using tools from *Kali Linux*. These attacks were labeled in the dataset by origin and destination IP filters, separating the traffic belonging to the attack machines from the normal lab network usage.

V. Automatic Threat Detection

Our proposed real-time architecture relies on machine learning algorithms to perform automatic threat classification. In Subections V-A, V-B, and V-C, we present the classification algorithms implemented to evaluate the architecture. We selected these algorithms because they are among the most popular for network security [10], [42]. In all methods, we perform the training with 70% of the dataset and the test with the remaining 30%. During the training phase, we perform tenfold cross-validation to avoid overfitting the data.

A. Decision Trees

In the decision tree algorithm, leaves represent the final class, and branches represent conditions based on the value of one of the input features. During the training part, the C4.5 algorithm determines a tree-like classification structure, based on the information entropy of each feature. The real-time implementation of the decision tree consists of if-then-else rules that generate the tree-like structure previously calculated. The results are presented in Section V-F, along with the other algorithms results. During the training phase, we set the following parameters: maximum number of splits as the number of samples minus one; minimum leaf size as one; and minimum number of branch node observations as ten observations. We chose these parameters because they ensured an excellent trade-off between the number of leaves and the tree depth. A deep tree with many leaves may result in high training accuracy but does not perform well in independent test sets.

B. Artificial Neural Networks

Artificial neural networks were originally inspired by the human brain, in which each neuron performs a small part of the overall processing, transferring the output to the next neuron, and achieving results from the combination of these subtasks. In classification neural networks, the final output represents a degree of membership for each class, and the output with the highest membership degree determines the predicted class. The training phase tunes the neural network adjusting the weight vectors Θ. These vectors determine the weight of each neuron connection. During the training phase, the input vectors are mapped into a predicted output vector and compared to the real output. The prediction errors are then minimized by the back-propagation algorithm, taking into account the error value induced by each parameter.

In the classification phase, each neural network layer computes the following equations:

$$z_{(i+1)} = \Theta_{(i)} a_{(i)} \quad (1) \qquad a_{(i+1)} = g(z_{(i+1)}) \quad (2) \qquad g(z) = \frac{1}{1+e^{-z}} \quad (3)$$

where $a_{(i)}$ is the vector that determines the output of layer i, $\Theta_{(i)}$ is the weight vector from layer i to layer $i+1$, and $a_{(i+1)}$ is the output of layer $i+1$. The function $g(z)$ is the *Sigmoid* function that plays an important role in the classification. For high values of z, $g(z)$ returns one and for low values $g(z)$ returns zero. Therefore, the output layer gives the degree of membership of each class, between zero and one. The trained neural network consisted of three layers, the input layer, the hidden layer, and the output layer. The input layer has 26 neurons and the output two since we classify each flow as a threat or normal. The hidden -layer size is equal to ten neurons. We chose the regularization parameter as one, to prevent data overfit.

C. Support Vector Machines

Support Vector Machines (SVMs) are binary classifiers based on the concept of a hyper-plane in a multidimensional space that splits different classes. An iterative algorithm minimizes an error function, finding the best hyper-plane separation. A kernel function defines this hyper-plane. This way, SVM finds the hyper-plane of maximum margin, that is, the hyper-plane with the biggest distance possible to both classes.

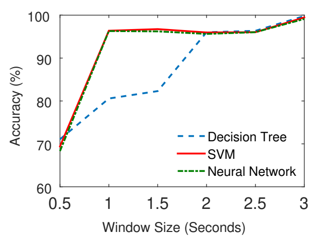
The classifier score for a sample x is the distance from x to the decision boundaries, which go from $-\infty$ to $+\infty$. The classifier score is given by:

$$f(x) = \sum_{j=1}^{n} \alpha_j y_j G(x_j, x) + b, \quad (4)$$

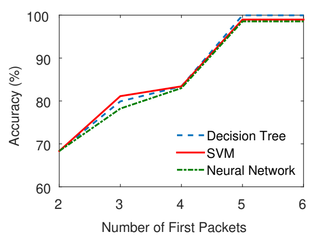
where $(\alpha_1, ..., \alpha_n)$ are the estimated parameters of SVM, and $G(x_j, x)$ is the used kernel. In this work, the kernel is linear, that is, $G(x_j, x) = x_j^{'} x$, which provides good performance with the minimum number of input parameters. We set the cost parameter as one to control the margin-violation penalty on observations, and to reduce overfitting. The initial coefficients, α_i, were randomly initialized.

D. Flow Parameter Tuning

We have to determine two critical parameters for our threat detection architecture: the window-time size for flows, and the number of initial packets needed to characterize a flow accurately. Small window-time sizes provide a faster threat detection but can compromise accuracy. Similarly, the fewer packets required to classify a flow, the quicker our algorithms can reach a decision. Figure 3 shows low-accuracy results for 0.5 second window size and two packets.This accuracy result, however, improves as the architecture gathers more information about the flows.



(a) Accuracy for different flow time window size.



(b) Accuracy for different number of first packets per flow.

Figure 3: Accuracy of decision tree, SVM, and neural network algorithms for the two flow approaches, all packets in a fixed time window size and first few packets of a flow.

We use the GTA/UFRJ dataset and the classification algorithms implemented to choose these parameters. Figure 3a shows the accuracy result for the approach with all packets gathered in different time windows. Both neural networks and SVMs provided good detection results, starting with a window size of just one second. At the same time, decision trees only achieved similar results with window sizes of at least two seconds. We choose a window size of just one second because that is enough information to classify the samples with the shortest possible time correctly.

We now turn our attention to selecting the number of packets to analyze. Figure 3b shows the accuracy for all three algorithms using the approach that analyzes the first few packets. Usually, the behavior of applications and threats is well defined early on, and as the result shows, five packets are enough to obtain high accuracy, near 99%. The first five-packets approach shows higher accuracy than the method that analyzes all packets for a one-second window size. This approach presents other advantages, such as shorter time to extract flow characteristics, greater robustness, and better scalability. Furthermore, it is robust to flooding attacks since there is no need to process a large number of packets for each flow.

E. Feature Selection and Principal Component Analysis

We perform dimensionality reduction to improve the efficiency of the proposed architecture for real-time threat detection. The aim is to improve the throughput, eliminating irrelevant features from the threat detection procedure. We also study possible correlations between two or more features, which can be combined into only one feature, reducing processing time. We perform dimensionality reduction with Principal Component Analysis (PCA), which transforms a group of possibly correlated variables into a group of weakly correlated variables that lie in orthogonal planes. This transformation takes into account the eigenvalues in a way that the component associated with the biggest eigenvalue represents more significant data variance. The algorithm sorts in variance order the other components to obtain the resulting matrix. We keep the components related to the higher eigenvalues because they have more relevant information, and we remove the components associated with low eigenvalues, reducing data dimensionality, and improving the processing time. It is important to remark that PCA does not consider the class label in the dataset, and therefore, can be used in both supervised and unsupervised learning.

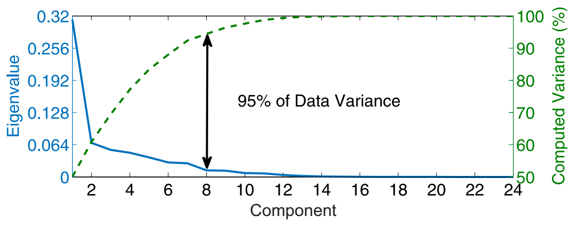


Figure 4: Eigenvalue for each flow feature. The eigenvalue associated to each of the transformed features is proportional to the data variance. The eight highest principal components represent 95% of the total data variance.

For both ways of combining packets into flows, the sum of the higher eight eigenvalues represents more than 95% of the total amount, as shown in Figure 4. In other words, the first eight features from the data calculated by the PCA linear transformation represent 95% of the total variance. Therefore, we select these eight components and discard the others that represent less than 5% of the total data variance, improving the processing time, which is critical in real-time applications.

F. Threat Classification Results

Tables I and II show the accuracy comparison for the first five packets of each flow, and the one-second time window approaches for both the GTA/UFRJ and NetOp datasets. The results show that using the first five-packets in the time window provides higher accuracy than the one-second time window for SVM and Neural Network classifiers. The accuracy improvement is due to the behavior of applications and threats, which are defined in the beginning, usually when the negotiation phase of applications happens. Both SVMs and Neural Networks perform well in all scenarios. As shown in Tables I and II, the accuracy for both these algorithms is higher than 95% for all scenarios, ensuring the efficiency of the proposed architecture to detect known threats.

Using the first five packets of each flow gives us better results than using all the packets in a one-second time window. Our proposal randomizes the time for checking the packets, and therefore, the attackers cannot easily simulate a legitimate use for a fixed time interval and then engage an attack.

Table I: Accuracy comparison for the 3 classification methods in the GTA Dataset.

	First Packets of Flow	**One Second Window**
Decision Tree	99.9%	80.6%
Neural Network	99.0%	96.0%
SVM	98.6%	96.3%

We further detail precision and recall for both legitimate and malicious traffic in Table III. Precision and recall are standard metrics for evaluating the results for a specific class. They are recommended metrics for evaluating intrusion detection

Table II: Accuracy comparison for the 3 classification methods in the NetOp Dataset.

	First Packets of Flow	One Second Window
Decision Tree	86.3%	92.8%
Neural Network	95.3%	95.1%
SVM	96.1%	95.8%

systems (they are similar to IDOC curves [24]), as they provide more meaningful results than ROC curves and the false positive vs. false-negative rates [24]. Since the primary goal of our threat detection architecture is to detect threats and trigger counter-measures, we show these metrics for the threat and normal classes, to indicate the number of true and false alerts that would trigger counter-measures.

Table III: Threat classification summary for all algorithms and datasets. We show precision and recall to evaluate the methods. The 1s stands for the one second time window approach for combining flows and the 5p stands for the first five packets in the period.

Algorithm	**Dataset**	**Threat as Positive**	
		Precision	**Recall**
Decision Tree	GTA 1s	80.1%	94.6%
	GTA 5p	99.9%	99.9%
	NetOp 1s	97.3%	94.8%
	NetOp 5p	99.1%	85.6%
Neural Network	GTA 1s	94.8%	98.8%
	GTA 5p	98.6%	99.8%
	NetOp 1s	97.5%	97.2%
	NetOp 5p	98.2%	96.6%
SVM	GTA 1s	95.4%	98.8%
	GTA 5p	98.0%	99.7%
	NetOp 1s	96.5%	98.9%
	NetOp 5p	99.6%	96.1%

For any given class, the precision is the fraction of correctly classified samples over all samples predicted to belong to such class. At the same time, the recall is the fraction of correctly classified samples over all samples that really belong to that class. As Table III shows, we achieve good results, usually above 90%, on both precision and recall for the normal class. Consequently, this ensures a low level of false alerts that would result in legitimate traffic block. Regarding the threat class, for the NetOp dataset, the precision results are lower, because the dataset has 85% of legitimate samples. Therefore, when evaluating the absolute number of normal samples classified as threat, they have a negative impact on the precision. Nevertheless, as shown in the results for the normal class, they do not result in many false alerts. Taking this into consideration, the most interesting measure to evaluate for the threat class is the recall that measures the threat-detection rate, that is, the number of correctly classified threats among all real threats in the datasets. The results show that the first five packets of each flow present a higher threat recall, therefore, a higher threat detection rate. Moreover, SVM is the most stable method, since, for all scenarios, the threat recall is above 87%.

G. *Anomaly Detection by Normal Distribution*

So far, we have used supervised-learning algorithms to detect previously seen attacks; however, protection against unknown attacks is essential to have a higher level of security in computer networks. Zero-day attacks are hard to detect since there is no previous information about the attack. Anomaly detection is capable of discovering these new attacks by identifying abnormal flows.

In the next two subsections, we present and compare two anomaly detection algorithms; (1) anomaly detection by distance to a Normal distribution, and (2) an anomaly detection algorithm based on Entropy metrics.

In the first method, we detect anomalies through the mean and variance from each feature of the normal samples of the training dataset. We identify anomalies when the distance from the sample feature to the mean is greater than a threshold times the variance in at least one of the features. We analyze the eight PCA transformed features.

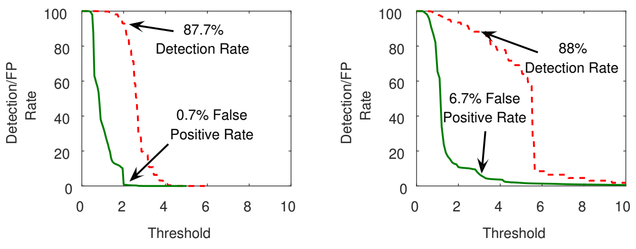
The real-time implementation requires the anomaly detection as the streaming data is arriving. The anomaly is detected if at least one of the following conditions is true for at least one feature j, taking into account the means μ_j and the variances σ_j^2 calculated in training:

$$X_j > \mu_j + threshold * \sigma_j^2 \quad (5) \quad X_j < \mu_j - threshold * \sigma_j^2 \quad (6)$$

The proposed architecture allows real-time anomaly training. Consequently, the algorithm becomes adaptive, which is fundamental for anomaly detection, since the network behavior may change in time. Therefore, when a new sample arrives and it is not detected as an anomaly by conditions (5) and (6), the parameters μ_j and σ_j^2 of each feature are updated, considering this new sample. The parameters of a normal distribution are expressed by:

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} X_j \quad (7) \quad \sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(X_j - \mu_j)^2 \quad (8)$$

Therefore, current values of the sum and the total of samples N are stored and incremented when a new sample arrives, considering each feature X_j. As a consequence, samples considered legitimate update the normal distribution parameters, ensuring adaptability.

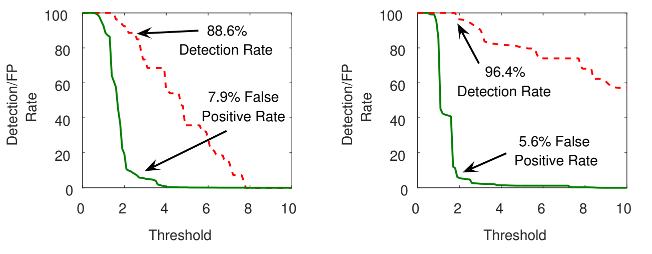


(a) First Five Packets of Flow. (b) One Second Time Window.

Figure 5: False positive and attack detection rates for the NetOp dataset according to the threshold.

Figure 5 shows the normal distribution anomaly detection and false-positive rates for the NetOp dataset. For the first packets of each flow approach, with a threshold value of 2.1, we achieve a remarkably low false positive rate of 0.7%, maintaining a good threat detection rate of 87.7%. For the one-second window approach, considering a threshold of 3, we obtain 6.7% and 88% false-positive and threat detection rates, respectively.

Figure 6 shows the results for the GTA/UFRJ dataset, considering different threshold values. We obtain the false positive rate using the normal test dataset and the threat detection rate using all the threats in the dataset. When choosing a low threshold value, the proposal detects almost all threats. Nonetheless, this comes with the cost of a high false-positive rate. On the other hand, a high threshold value results in fewer false-positives, but also a lower threat detection rate. With a threshold value of two, the false positive rate is 5.6%, and the detection rate 96.4% for the one second time window approach. With a threshold of 2.7, these rates are 7.9% and 88.6% for the first five packets of each flow approach. The value for the threshold parameter depends on the application and considers a trade-off between the false-positive and attack detection rates.



(a) First Five Packets of Flow. (b) One Second Time Window.

Figure 6: False-positive and attack detection rates for the GTA/UFRJ dataset according to the threshold. The lower the threshold, more attacks are detected, but also higher is the false positive rate.

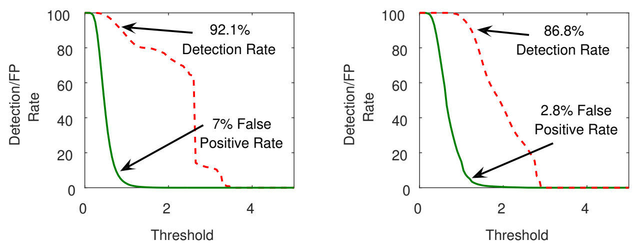
H. Anomaly Detection by Entropy Time Series

We also implement an anomaly detection algorithm by analyzing the entropy value of a sliding window of flows. The sample entropy indicates the degree of concentration or dispersion of a characteristic. It is calculated as follows:

$$H(X) = -\sum_{i=1}^{N}(\frac{n_i}{S})\log_2(\frac{n_i}{S}), \quad (9)$$

where S is the total number of observations and n_i is the number of observations within the range i of values, and N is the number of ranges. When one range concentrate all values, $H(X)$ is equal to zero and when each value is in a different range i, the value of $H(X)$ is $\log_2(N)$. Therefore, given a series of observations X, the sample entropy summarizes the level of concentration in one single value.

To detect anomalies using the sample entropy value, we define a sliding window of 40 flows and calculate the value of $H(X)$ for each of these windows, generating the time series. In the training phase, we calculate the histogram of all normal samples of the training dataset and determine 30 ranges of equally distributed entropy values. Another parameter specified in the training phase is the range of most samples. We observed that the normal traffic entropy values tend to concentrate together, and that usually, the most frequent value is in the middle of this concentration. Thus, we propose to detect anomalies by defining a threshold that limits the accepted distance of the entropy $H(X)$ to the most frequent range. The architecture can update the most frequent value online, adapting to different network behaviors.

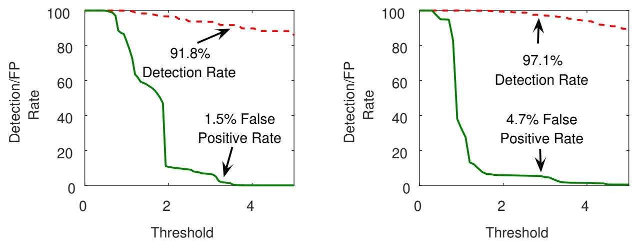


(a) First Five Packets of Flow. (b) One Second Time Window.

Figure 7: False positive and attack detection rates for the GTA/UFRJ dataset according to the entropy threshold. The threshold represents the distance to the range that has the most entropy samples.

Figure 7 shows the results for different threshold values considering the GTA/UFRJ dataset. For the first five packets of each flow approach, with a threshold of 0.8, the threat detection rate is 92.1%, and the false-positive rate is 7%. For the one-second window and the threshold value of 1.3, these values are 86.8% and 2.8%. Once again, we can determine this threshold, considering the trade-off between threat detection and false-positive rates.

Figure 8 shows the entropy anomaly detection results for the NetOp dataset. For the first packets of each flow approach, the threat detection rate is very high, 91.8%, and the false-positive rate is low, 1.5%. The result for the one second time window is as good as the other approach, with false-positive and detection rates of, respectively, 4.7% and 97.1% with a threshold of 3. The results for this method shows an excellent trade-off in all scenarios.



(a) First Five Packets of Flow. (b) One Second Time Window.

Figure 8: False positive and attack detection rates for the NetOp dataset according to the entropy threshold. The threshold represents the distance to the range that has the most entropy samples.

VI. The Threat Prevention Architecture

We are now ready to have all our algorithms in a combined threat prevention architecture that analyzes traffic online, in batch mode, and that, also, provides fast attack response algorithms using the programmability of SDNs. We classify intrusion detection systems in two modes: on-path and off-path. In the on-path mode, we accomplish the analysis in the path from the source to the destination. In the off-path mode, the packets are mirrored and sent to an investigation that will determine whether the packet is part of a threat. The great advantage of the on-path mode is the ability to block traffic actively. This approach, however, introduces latency problems, considering that we add the processing time to analyze before the forwarding procedure. Moreover, the off-path mode allows gathering information from other sources. Thus, the system can correlate essential data to improve detection accuracy. On the other hand, the off-path mode needs an additional infrastructure to carry both traffic mirroring, and threat blocking.

We perform threat detection based on information from the first five packets of each flow (as discussed in the previous section). Whenever an undefined flow arrives at an SDN switch, it forwards a message to the controller, which then installs a rule with two actions, the first one forwarding the flow to its destination and the second one replicating the flow to an analysis machine. Meanwhile, the analysis machine keeps track of the number of packets for each flow that it is analyzing. When that number reaches five packets, this machine sends the flow information to the threat detection application and also a message warning to the controller, informing the accomplishment of packets analysis of a particular flow. We define flow as all packets from the same IP source and to IP destination, and the analysis machine extracts 26 flow features and publishes it in the `Kafka` message broker so that the processing core of the Detection Architecture gets the information and determines whether it is a threat.

Once the controller receives the message indicating the flow analysis completion, it lists all flows with the source and destination IP of the specified flow and removes the mirroring rule. The controller keeps the routing action to the flow destination, which preserves network operation. Thus, after analysis, the flow is no longer forwarded to the analysis machine, making the process robust because the analysis machine is protected against flooding attacks and has a greater ability to analyze flows when compared to the alternative of analyzing all packets. It is important to remark that the controller installs and removes the mirroring action periodically to increase network security. Therefore, after removing the mirroring action, it reinstalls again after the flow timeout set in the controller, which forces the periodical check of the flows. This timeout is randomly chosen within a range of values, to avoid an attacker that tries to bypass our system by sending legitimate flows during the beginning of the analysis.

Figure 9 shows an example network analyzed and protected by the proposed Threat Detection and Prevention Architecture. All SDN switches have an interface to which traffic is duplicated and sent to an analysis machine. The controller is responsible for installing the mirroring rule and for removing it when the machine accomplished the reception of the five required packets. The capture module, composed of the analysis machines, then forwards the flow characteristics to the detection application, which utilizes machine learning algorithms in conjunction with stream processing in real-time to do its analysis. If a threat is detected, the detection application sends an alert to the controller, which then blocks the attacker source IP in all switches.

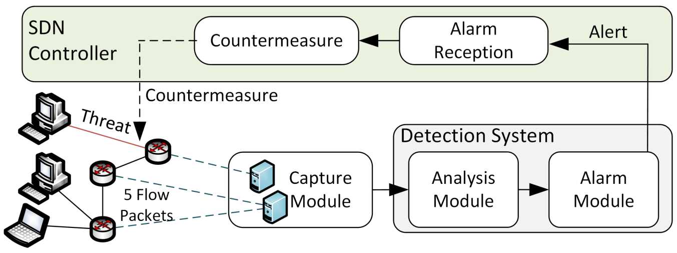


Figure 9: Example of network monitored by the proposed architecture. The network controller installs the mirroring rules for the capture module that sends traffic information to the detection system. Through alerts from the detection system, the network controller can block the threats.

A. *Defense strategy against attacks with Spoofed Source IP*

The source IP blocking rule is ineffective against attacks that falsify the source IP address of their packets to deceive defense systems. Attacks with spoofed source IP are even more critical in Software Defined Networks. Besides the attack damage, they overload the controller to set new flows every time the source IP changes, also generating a large number of entries in the switches flow table.

We propose a strategy against spoofed IP attacks based on a sequence of alerts and marking the path of flows. The intuition behind this concept is as follows: if the controller installed a blocking rule against an attack, and even then, an alarm arrived soon after, it may indicate that the blocking rule was not effective. Therefore, the controller keeps track of time between the reception of alert and notices when two alerts arrive in a short time. When this happens, the controller begins to suspect that the packet belonging to the attack has a spoofed IP. Due to this suspicion, the controller starts to map the path of all network flows for a specified period. This mapping is possible since, in SDNs, the controller has a global network view and knows the entire network topology. Then, if a third warning reaches the controller, in addition to the traditional source IP blocking rule, the controller also finds out in which switch port the attacker traffic enters the network and blocks this port. Thus, the controller instantiates a blocking rule of the traffic from the attacker TCP port, in addition to the IP blocking rule. Here we block the traffic from the port, considering that the intruder is in the SDN network. Nevertheless, other actions are applicable, such as limiting the bandwidth of the suspected attacker or transferring the traffic to a honeypot. Therefore, if an attacker spoofs its source IP, after three warnings, the attack traffic is blocked. Asking the controller to monitor the path of all flows only occurs when two alerts happen in a short period.

On the other hand, after a period without receiving any new alert, the process is undone, avoiding the waste of resources. Another consideration is that instead of an attack with spoofed IP, it could be a distributed attack. Against distributed attacks, it is essential to block all sources, and thus the blocking rules are effective when blocking traffic from both the inbound network interface and the source IP.

```
input  : Incoming Alerts from the Detection Methods
output: Blocked source IP or incoming port
Initialize threshold, status;
while True do
    $wait(alert)$;
    $block(alert.sourceIP)$;
    switch status do
        case normal do
            if
              $alert.time - lastAlert.time < threshold$
              then
                $mapSourcePort(start)$;
                $status = monitoring$;
            end
        end
        case monitoring do
            if
              $alert.time - lastAlert.time > threshold$
              then
                $mapSourcePort(stop)$;
                $status = normal$;
            else
                $block(sourcePort(alert.sourceIP))$;
            end
        end
    end
    $lastAlert = alert$;
end
```

Algorithm 1: Defense strategy against attacks with spoofed source IP

Algorithm 1 summarizes the defense strategy against spoofed IP addresses. The controller has two states: normal and monitoring. In the normal state, there is no suspicion that an attack with spoofed IP is occurring. In the monitoring state, there are suspicious activities, and the controller maps the source port of every incoming flow, using its global network view. The controller changes the status based on the time difference between two consecutive alerts. When this difference is above a threshold, the controller either keeps the monitoring status or goes back to the normal status. Similar behavior occurs when this difference is below the threshold, however, the controller changes to or keeps the monitoring status. An essential aspect of the proposed strategy is that we only map the flows under spoofed IP suspicion, and we return to the normal status to avoid a waste of resources. The first step of the algorithm once an alert arrives is to block its source, therefore, preventing the network against distributed threats by blocking each one of them. Here, we implemented the action of blocking the source port when detected a threat that probably is spoofing its IP. Other counter-measures, however, are applicable, such as reducing the bandwidth or forwarding to a honeypot or network filter.

B. Traffic Monitoring and Threat Blocking

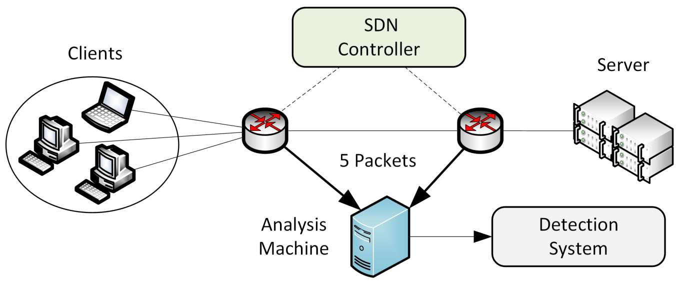
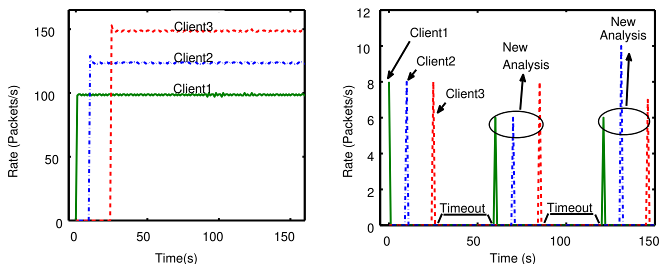


Figure 10: Experimental network topology used for the five packets scheme and the malicious traffic block.

In this platform, we use Xen hypervisor for virtualization and OpenFlow for traffic forwarding. Figure 10 shows the constructed topology for the experiments, which consists of three client machines and a server machine. The packet forwarding is accomplished by OpenvSwitch switches that are controlled by an application programmed in the POX controller. Furthermore, an analysis machine characterizes flows using `Bro`. The connection between switches and the analysis machine requires a Generic Routing Encapsulation - (GRE) tunnel, which encapsulates the packet and sets the address of the analysis machine as a destination. The analysis machine parses the packets and then analyzes them. We perform the experiments on an Intel Xeon X5690 server with 24 processing cores, each with a frequency of 3.47 GHz clock and with 48 GB of RAM.

The results of the first experiment, shown in Figure 11, aim to display the operation of the traffic mirroring rule and the subsequent end of this rule after the analysis of the first five packets. In this experiment, the three clients send traffic at a constant rate to the server machine. Figure 11a shows the traffic received by the server machine. The results show that the proposed scheme does not affect communication. Figure 11b shows the packet rate received by the analysis machine, which sends a message to the network controller after capturing the five packets needed to characterize the flow. Even though the sending rate is much higher, the analysis machine receives a few packets. The reason the analysis machine receives a little more than five packets is the time required to inform the controller to undo the mirroring rule. Besides, this figure shows that flows are periodically analyzed accordingly to the flow timeout set in the controller. In this experiment, we set the flow timeout within 60 seconds. Therefore, the analysis machine receives the packets of each client every minute.

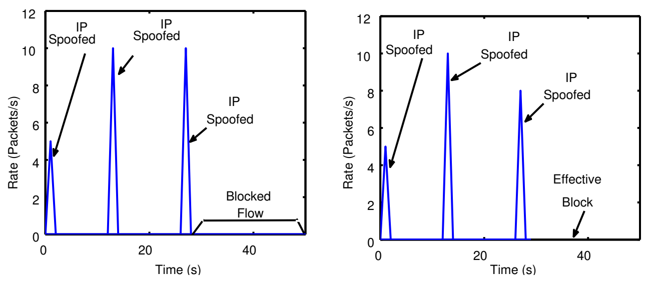
Two crucial aspects of this proposal are the time required to characterize the flow and the ability to increase the number of flows to be analyzed. As the machine only needs five packets to characterize flows, there is no need to wait until the end of the flow or connection to send the information to the Threat Detection Architecture, which results in shorter



(a) Packet rate received by the server machine. (b) Packet rate received by the analysis machine.

Figure 11: Network operation when three machines communicate with the server. The server receives traffic without being affected by the proposed scheme, while the analysis machine only receives the first packets of each flow.

detection times and, thus, a faster threat block. Furthermore, the analysis machine does not process all packets and then is immune to flooding attacks. Therefore, for this reason, the analysis machine can receive a much larger number of flows, ensuring robustness and potential for scalability.



(a) Packet rate of attacker received by the server machine. (b) Packet rate of the attacker received by the analysis machine.

Figure 12: Architecture operation under an attack that spoofs the source IP address. Blocking rules cease to be effective when the IP is spoofed, however after the identification of the interface through which the attack enters the network, the attack is effectively blocked.

The second experiment shows one of the client machines attacking the server. The attacker machine performs a threat that also spoofs the source IP to avoid detection. Similar to the first experiment, Figure 12 shows the traffic received by the server and the analysis machine. Our system immediately installs on all switches a rule blocking the source IP after the attack started, and the analysis machine detected it, sending the information to the detection application. This procedure blocks the malicious traffic received by the server, as shown in Figure 12a. Nevertheless, when the attacker changes the IP for the first time, the flow is once again analyzed and blocked, as shown in Figure 12b. Nonetheless, this time, when the controller receives the alarm, it starts to mark the path of all flows, so it can map in which port each flow enters the network. The system generates a new alert once more after the analysis, around 28 seconds, because the attacker changes the source IP. When the controller receives this second alert, it blocks both the IP source and the port in which this flow enters the network. Here we choose to block the traffic from the TCP port in which the attacker enters the network. However, other policies are applicable, such as redirecting the traffic to a filter out the malicious flow. From this point on, when the attacker changes once more the source IP address, the traffic is not analyzed again since it is blocked in as near to its origin as possible.

VII. Conclusion

This paper proposes a fast and accurate Threat Detection and Prevention Architecture. Machine learning algorithms perform threat detection using combined with stream processing. We implement five algorithms to detect known and unknown attacks and obtain high accuracy in classification, higher than 95%, and an excellent trade-off between threat detection and false-positive rates in anomaly detection. The results show that our architecture handles well both known attacks and detects zero-days attacks even without any previous information, only based on normal network behavior. Another contribution of this work is the creation of a security dataset to evaluate our proposal. Moreover, we also use another dataset with real data collected from broadband users of one of the most important network operators in Brazil. Both datasets contain real traffic data instead of simulation results. We measure the processing throughput of the proposed Threat Detection Architecture, obtaining a threat detection time of about four microseconds per sample, enabling prompt counter-measures against attackers and showing the scalability of our architecture. The proposed system achieves a quick threat detection time thanks to the stream processing technology combined with machine learning algorithms in the lambda architecture.

Besides, we propose a scheme based on Software Defined Networking and the periodic analysis of the first five packets of each flow in our Threat *Prevention* Architecture. The scheme ensures a quick detection since it does not have to wait until the end of a flow to characterize it. The Threat Prevention Architecture mirrors the traffic to sensors spread around the network. Therefore, the architecture does not add any delay to user communication. Moreover, the scheme performs an effective threat block, even in scenarios in which the attacker uses spoofed IP addresses. The controller receives alerts from the threat detection application and installs rules to block the attacker source IP. However, when an attacker changes its IP, the architecture detects it, based on the time difference between the alerts and maps the source of the attack to effectively block the threat. The proposed scheme is robust since both the controller and the analysis machine are protected against flooding attacks since not all packets are mirrored to the analysis machine and thus prevents overloading the controller. The proposed architecture scales well since its threat detection time is low, due to the stream processing core that improves its throughput when the parallelism increases. Furthermore, the schema that analyzes only a few packets of each flow prevents an attacker to flood sensor elements.

Acknowledgment

This work was supported by INCT of the Future of the Internet, CNPq, CAPES, FAPERJ, and FAPESP. The authors would like to thank Renato Souza Silva for the contribution with the Network Operator Dataset.

References

[1] S. Suthaharan, "Big data classification: Problems and challenges in network intrusion prediction with machine learning," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 4, pp. 70–73, 2014.

[2] A. A. Cardenas, P. K. Manadhata, and S. P. Rajan, "Big data analytics for security," *IEEE Security Privacy*, vol. 11, no. 6, pp. 74–76, Nov. 2013.

[3] N. C. Fernandes and O. C. M. B. Duarte, "Xnetmon: A network monitor for securing virtual networks," in *IEEE International Conference on Communications ICC*, 2011, pp. 1–5.

[4] S. Thangavel and S. Kannan, "Detection and trace back of low and high volume of distributed denial-of-service attack based on statistical measures," *Concurrency and Computation: Practice and Experience*, vol. n/a, no. n/a, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5428

[5] Kaspersky, "Kaspersky ddos intelligence report for q1 2016." Kaspersky Lab., Tech. Rep., 2016. [Online]. Available: https://securelist.com/kaspersky-ddos-intelligence-report-for-q1-2016/74550/

[6] E. Fenil and P. Mohan Kumar, "Survey on ddos defense mechanisms," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 4, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5114

[7] P. Clay, "A modern threat response framework," *Network Security*, vol. 1, no. 4, pp. 5–10, 2015.

[8] M. D. D. Moreira, R. P. Laufer, N. C. Fernandes, and O. C. M. B. Duarte, "A stateless traceback technique for identifying the origin of attacks from a single packet," in *IEEE International Conference on Communications (ICC)*. IEEE, 2011, pp. 1–6.

[9] R. P. Laufer, P. B. Velloso, D. de Oliveira Cunha, I. M. Moraes, M. D. D. Bicudo, and O. C. M. B. Duarte, "Towards stateless single-packet ip traceback," in *IEEE Conference on Local Computer Networks LCN'2007*. IEEE, 2007, pp. 548–555.

[10] A. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys Tutorials*, vol. 18, no. 99, pp. 1–26, 2015.

[11] S.-Y. Ji, B.-K. Jeong, S. Choi, and D. H. Jeong, "A multi-level intrusion detection method for abnormal network behaviors," *Journal of Network and Computer Applications*, vol. 62, pp. 9–17, 2016.

[12] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, pp. 217–228, Aug. 2005.

[13] G. Fernandes, L. F. Carvalho, J. J. P. C. Rodrigues, and M. L. Proença, "Network anomaly detection using IP flows with principal component analysis and ant colony optimization," *Journal of Network and Computer Applications*, vol. 64, pp. 1–11, 2016.

[14] F. Palmieri, U. Fiore, and A. Castiglione, "A distributed approach to network anomaly detection based on independent component analysis," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 5, pp. 1113–1129, 2014.

[15] H. Ringberg, A. Soule, J. Rexford, and C. Diot, "Sensitivity of PCA for traffic anomaly detection," in *ACM SIGMETRICS*, 2007, pp. 109–120.

[16] A. A. Amaral, L. de Souza Mendes, B. B. Zarpelão, and M. L. P. Junior, "Deep IP flow inspection to detect beyond network anomalies," *Computer Communications*, vol. 98, pp. 80–96, 2017.

[17] C. P and S. K. M. P. D, "Real-time anomaly detection using parallelized intrusion detection architecture for streaming data," *Concurrency and Computation: Pratice and Experience*, vol. 32, no. e5013, Jan. 2020.

[18] E. Villar-Rodríguez, J. Del Ser, A. I. Torre-Bastida, M. N. Bilbao, and S. Salcedo-Sanz, "A novel machine learning approach to the detection of identity theft in social networks based on emulated attack instances and support vector machines," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 4, pp. 1385–1395, Mar. 2016. [Online]. Available: http://dx.doi.org/10.1002/cpe.3633

[19] B. Li, S. Zhang, and K. Li, "Towards a multi-layers anomaly detection framework for analyzing network traffic," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 14, p. e3955, 2017.

[20] H. Bostani and M. Sheikhan, "Hybrid of anomaly-based and specification-based IDS for internet of things using unsupervised OPF based on MapReduce approach," *Computer Communications*, vol. 98, pp. 52–71, 2017.

[21] K. Singh, S. C. Guntuku, A. Thakur, and C. Hota, "Big data analytics framework for peer-to-peer botnet detection using random forests," *Information Sciences*, vol. 278, pp. 488–497, 2014.

[22] E. Viegas, A. Santin, A. Bessani, and N. Neves, "Bigflow: Real-time and reliable anomaly-based intrusion detection for high-speed networks," *Future Generation Computer Systems*, vol. 93, pp. 473 – 485, 2019.

[23] M. A. Lopez, D. M. F. Mattos, O. C. M. B. Duarte, and G. Pujolle, "Toward a monitoring and threat detection system based on stream processing as a virtual network function for big data," *Concurrency and Computation: Practice and Experience*, vol. 1, no. 1, pp. 1–17, 2019.

[24] A. A. Cardenas, J. S. Baras, and K. Seamon, "A framework for the evaluation of intrusion detection systems," in *IEEE Symposium on Security and Privacy (SP'06)*, May 2006, pp. 15 pp.–77.

[25] W. Lee, S. J. Stolfo, and K. W. Mok, "Mining in a data-flow environment: Experience in network intrusion detection," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1999, pp. 114–124.

[26] P. Sangkatsanee, N. Wattanapongsakorn, and C. Charnsripinyo, "Practical real-time intrusion detection using machine learning approaches," *Computer Communications*, vol. 34, no. 18, pp. 2227 – 2235, 2011.

[27] M. Amini, R. Jalili, and H. R. Shahriari, "RT-UNNID: A practical solution to real-time network-based intrusion detection using unsupervised neural networks," *Computers & Security*, vol. 25, no. 6, 2006.

[28] M. A. Lopez, D. M. F. Mattos, and O. C. M. B. Duarte, "An elastic intrusion detection system for software networks," *Annals of Telecommunications*, pp. 1–11, 2016.

[29] M. A. Lopez and O. C. M. B. D. Duarte, "Providing elasticity to intrusion detection systems in virtualized software defined networks," in *IEEE International Conference on Communications ICC'15*, Jun. 2015, pp. 7120–7125.

[30] Y.-D. Lin, P.-C. Lin, C.-H. Yeh, Y.-C. Wang, and Y.-C. Lai, "An extended SDN architecture for network function virtualization with a case study on intrusion prevention," *IEEE Network*, vol. 29, no. 3, pp. 48–53, 2015.

[31] P. K. Sharma, S. Singh, and J. H. Park, "Opcloudsec: Open cloud software defined wireless network security for the internet of things," *Computer Communications*, vol. 122, pp. 1 – 8, 2018.

[32] C. Vicentini, A. Santin, E. Viegas, and V. Abreu, "SDN-based and multitenant-aware resource provisioning mechanism for cloud-based big data streaming," *Journal of Network and Computer Applications*, vol. 126, pp. 133 – 149, 2019.

[33] A. G. P. Lobato, M. A. Lopez, I. J. Sanz, A. A. Cardenas, O. C. M. B. Duarte, and G. Pujolle, "An adaptive real-time architecture for zero-day threat detection," in *IEEE International Conference on Communications - ICC*, May 2018, pp. 1–6.

[34] M. Rychly, P. Koda, and P. Smrz, "Scheduling decisions in stream processing on heterogeneous clusters," in *Eighth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, Jul. 2014, pp. 614–619.

[35] N. Marz and J. Warren, *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2013.

[36] M. Andreoni Lopez, A. Lobato, and O. C. M. B. Duarte, "A performance comparison of open-source stream processing platforms," in *IEEE GLOBECOM*, Washington, USA, Dec. 2016, pp. 1–6.

[37] Z. Cheng, J. Caverlee, and K. Lee, "You are where you tweet: A content-based approach to geo-locating twitter users," in *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, ser. CIKM'10. ACM, 2010, pp. 759–768.

[38] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham *et al.*, "Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation," in *Proceedings of DARPA Information Survivability Conference and Exposition. DISCEX'00.*, vol. 2. IEEE, 2000, pp. 12–26.

[39] M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications*. IEEE, 2009, pp. 1–6.

[40] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *IEEE Symposium on Security and Privacy (SP)*. IEEE, 2010, pp. 305–316.

[41] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Computers & Security*, vol. 45, pp. 100–123, 2014.

[42] Q. Liu, P. Li, W. Zhao, W. Cai, S. Yu, and V. C. Leung, "A survey on security threats and defensive techniques of machine learning: a data driven view," *IEEE access*, vol. 6, pp. 12 103–12 117, 2018.