An Accurate Threat Detection System through Real-Time Stream Processing

Antonio Gonzalez Pastana Lobato, Martin Andreoni Lopez, Otto Carlos M. B. Duarte
Universidade Federal do Rio de Janeiro - GTA/COPPE/UFRJ - Rio de Janeiro, Brazil
Email: {antonio, martin, otto}@gta.ufrj.br

***Abstract*—The late detection of security threats causes a significant increase in the risk of irreparable damages, disabling any defense attempt. All attacks leave detectable traces, even though most of them are complex and very hard to analyze. This paper proposes a real-time threat detection system based on stream processing and machine learning algorithms. The system architecture combines the advantages of real-time streaming with the batch processing over a historical database and it does not require any intervention of security specialists. The proposed system allows both attack classification and anomaly-based detection of known and zero-day attacks. The system was developed and evaluated with a dataset constructed by the capture of legitimate and malicious network traffic. Results show that the proposed system presents an accurate threat detection with low processing time, allowing prompt defense strategies.**

I. Introduction

The threats and security attacks are currently spread and tend to increase significantly in the future with the Internet of Things (IoT), once more than 80 billion of devices are estimated to be interconnected by 2025 [1]. This scenario displays a high management and protection complexity in communication networks, with several challenges in security and data privacy. The billions of devices generate a big amount of data streams, which needs to be managed, processed, transferred and stored in a secure real-time way. Besides, velocity, volume, and variety of big data are factors that increase the number of vulnerabilities.

The threat detection time is of the essence to maintain security in communication systems. The effective threat detection demands monitoring, processing, and management of big data, which allows information extraction from traffic characterization. However, detecting threats in big data requires the development of modern analytic techniques. Current security systems, such as Security Information and Event Management (SIEM), designed to gather data and analyze it in a single point, are not effective, since 85% of network intrusions are detected weeks after they happened [1]. Moreover, the reaction to threats after its detection is very slow, taking an average time of 123 hours. In addition, data leaking detection time is 206 days [1]. Therefore, the long threat detection time makes any kind of defense unfeasible. Analytic techniques for real-time stream processing enables the immediate analysis of different kinds of data and, consequently, the threat detection. Another usual problem of current security systems is the huge amount of alarms. Most of these alarms are false positives, which are ignored. This scenario, however, overwhelms the security analysts and real positives attacks are also ignored.

This paper proposes and implements an accurate real-time threat detection system, using open source platforms. The integrated system allows big data analysis in a stream processing manner. The proposed system uses machine learning for both attack classification and anomaly detection. Furthermore, we create a dataset with labeled classes for the system evaluation, containing normal network usage and several attacks from packet captures, abstracted in flows. The system architecture is based on the concept of the lambda architecture [2], which combines traditional batch processing over a historical database with real-time stream processing analysis. Therefore, the proposed system is able to detect both known and zero-day attacks through automatized classification and anomaly detection methods. A system prototype was developed and evaluated. The results show high accuracy of the system to detect threats. Furthermore, the proposed system rapidly detects security threats, the stream processing is associated with the historical database processing to adapt the detection algorithms in real-time. This results in a system with a low number of false positives, enabling defense techniques.

The rest of the paper is organized as follows. Section II discuss related work. The proposed system is presented in Section III. The dataset to evaluate the system is presented in Section IV. In Section V the threat detection methods and results are discussed. Finally, Section VI concludes the work.

II. Related Work

Machine learning techniques can be either supervised or unsupervised, depending on whether the dataset is labeled or not. The supervised analysis classifies attacks. Among the most well-known classification techniques are neural networks, decision trees, and Support Vector Machines (SVM) [3]. In the unsupervised analysis, there is no information regarding the class to which each sample belongs. Pattern detection applies this kind of analysis. However, none of those techniques provides real-time responses.

Snort and Bro are the most popular open source real-time intrusion detection tools [4]. Snort only uses signature based intrusion detection, failing to detect attack variations and requiring frequents updates in its signature database. Bro, on the other hand, is a framework for anomaly detection, whereas the user must create its own applications. These tools improve attack prevention and can be used to prevent intrusion in virtual networks [5].

Another important aspect approached by researches is the creation of datasets to analyze, evaluate, and compare different

threat detection systems. Sperotto *et al.* [6] created a dataset with over 14 million of labeled flows. A number of other works also create their own dataset to evaluate their proposals [7], [8]. Sangkatsanee *et al.* classify attacks using different algorithms, while Morteza Amini *et al.* 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.

Johnson and Lazos [9] propose anomaly detection by aggregating IP flows. The results, however, are not analyzed for real-time scenarios. Du *et al.* [10] applies a stream processing platform to detect anomalies. Nevertheless, they only analyses incoming packet rates and therefore only detects peaks that not necessarily correspond to anomalies, obtaining possibly a high false positive rate. Zhao *et al.* [11] also use a stream processing platform to detect network threats, but their results are preliminary.

Unlike the previously cited papers, this work proposes the use of open source platforms in a specific architecture that allows stream processing and analysis in real-time based on the support of a historical database. Our system provides real-time accurate detection of known and zero-day attacks through automatized classification and anomaly detection methods.

III. The Proposed Threat Detection System

Analysis of massive statistical data usually employs batch processing. However, this technique produces high latency, with responses in the order of tens of seconds, while a great number of critical applications require real-time processing, with responses within a second [12]. On the other hand, stream processing techniques analyze massive unbounded data that are continuously generated. Both these paradigms, batch and stream processing, are combined in the lambda architecture to analyze big data in a real-time manner [2].

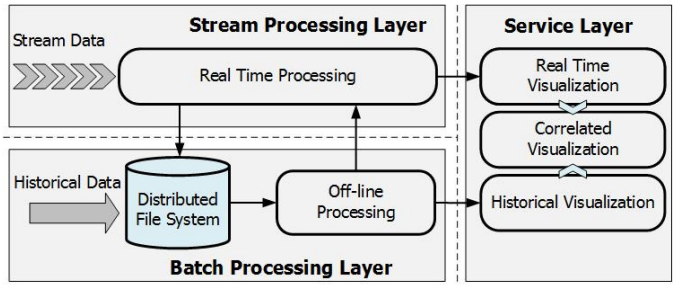


Figure 1: The three-layered lambda architecture, which combines stream with batch processing: stream processing, batch processing, and service layers.

The proposed system is based on the lambda architecture, shown in Figure 1, that allows the real-time manipulation and analysis of massive amounts of information. 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 analyses 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 by analytic data to the user. Therefore, the lambda architecture goal is to analyze, accurately and in real-time, streaming data, even with its ever-changing incoming rate to obtain results in real-time based on historical data.

The data analysis is divided in three steps: capture, normalization and processing. First, the system captures data. Then, every acquired information is sent to the normalization process, in which the data are formatted and enriched at high processing rates with external parameters, such as geographic information, correlations, and so on. Hence, the information about the attacks are of higher quality. Under attack situations, the correlation and enrichment of data from different sources make it possible to determine who the attacker is, his location, the data leaks, the destinations to which the attacker send the leaked data, among other things. Lastly, the normalized data are processed in real-time, gathering the required security patterns. In this process, once the system obtains the results, they are visualized and stored.

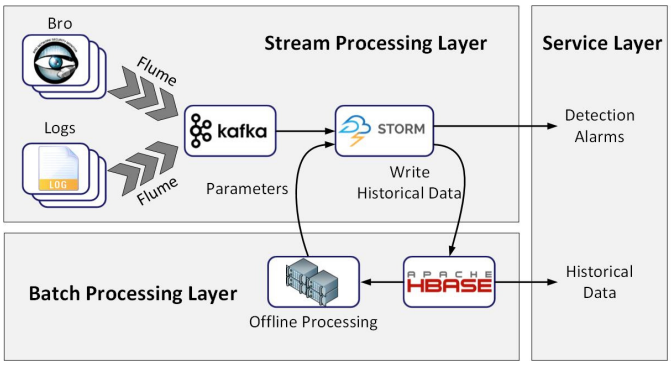


Figure 2: Proposed system architecture for real-time stream processing and also historical data batch processing using open source tools.

The threat detection prototype of the proposed system integrates a set of open source tools. Figure 2 shows how the open source tools are interconnect to compose the proposed system architecture. The system captures data from different sources. These sources are application and operating system logs, besides network traffic collected by `Bro` at different network locations. `Bro` is a real-time traffic analysis framework that has its own network oriented programming language, making flow abstraction easier. Therefore, in the proposed system, `Bro` characterizes the network flows, synthesizing the packets information and grouping then in time windows.

The system aims to ensure security in several network components. Thus, the data capture is distributed in different network locations and, then, the data is grouped to be processed. Both flow information generated by `Bro` and application logs are transported to the analysis location by the Apache `Flume` tool, which acts as a tunnel connecting the data sources to the system processing element. Since the network generates captured data at different rates, an overload

may occur at the stream processing, resulting in data loss. To avoid that, the captured data are received by the tool Apache `Kafka` at the analysis location. Basically, `Kafka` is a message broker that works as a publish/subscribe service and acts like a buffer to the processing tool, adapting different generation and processing rates. `Kafka` abstracts the message flow into topics. Producers then write their data in topics from which the consumers read these data. Apache `Storm` is the stream processing core of the system to detect threats. `Storm` offers a distributed fault tolerant stream processing framework. Thereby, `Storm` process data in memory, ensuring low latency in real-time. The streams are processed in a topology, that is, a Direct Acyclical Graph (DAG) composed of input elements, *Spouts*, and processing elements, *Bolts*. The application can define the parallelism of the *Spouts* and *Bolts*, in a way that multiple stream samples can be processed simultaneously.

Once the system extracts all the analytics from the stream data, the results are stored in a dynamic database to a posterior analysis. The proposed system database is Apache `HBase` that stores massive amounts of spread data and real- time access. The stored data have the information gathered during the threat detection and can be processed off-line to calculate parameters to be used in the real-time model. At this point, there is a feedback, since the parameters calculated off-line with historical data adjust the processing model in real-time. Thus, the system has an adaptive characteristic, because the parameters are updated, adjusting to new network use patterns.

IV. Security Dataset Creation

Only a few network security datasets are available to evaluate defense mechanisms. The main reason not to provide security data is privacy, since traffic data may contain confidential information. The two best known available datasets are DARPA [13] and KDD 99 [14]. TCP/IP traffic and operating system data collected from a simulated computer network compose the DARPA dataset. While collecting the data, attacks were also simulated and labeled in the dataset. The KDD 99 consists of a selection and grouping of DARPA features. However, these datasets present several limitations [15]. One of them is that the traffic does not correspond to a real network scenario, since it was simulated. Besides that, there are redundant data, which affects the algorithms results. Another issue is that these datasets are over 15 years old and does not represent current attack scenario [16].

A contribution of this work is the creation of a dataset with real network traffic to evaluate the proposal. A dataset is elaborated through the packet capture in computers from our lab, GTA at Federal University of Rio de Janeiro, containing both normal traffic and real network threats. After the packet capture, data from the header, grouped in a time window, generated flow data. We define a flow as a sequence of packets from the same IP source to the same IP destination. Each flow has 24 features, generated by TCP/IP header data, as TCP, UDP and ICMP packet rate, number of source and destination ports, number of each TCP flag, among others. The analysis of packet header information detects two threat classes: Denial of Service (DoS) attacks and probe. Therefore, we elaborate the dataset with several attacks from both these classes. Altogether, the dataset contains seven types of DoS and nine types of probe. The DoS attacks are *ICMP flood*, *land*, *nestea*, *smurf*, *SYN flood*, *teardrop*, and *UDP flood*. The different types of probe 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 perform the threats using tools from the *Kali Linux* distribution, which aims to test computer system security. These attacks were labeled in the dataset by origin and destination IP filters, separating the traffic belonging the attack machines from the rest.

Altogether, around 95 GB of packet capture data were collected, resulting in 214,200 flows between normal and malicious traffic. In order to do attack classification, the dataset was divided in 70% for training and 30% to evaluate the performance of the implemented algorithms. This procedure results in statistics that are determined with new data, not analyzed in the training phase, as usually done in scientific works in the area. For the anomaly detection, the training is performed with 70% of legitimate flow data to determine normal behavior. The other 30% are used to determine false positives rate and the attack data are used to calculate the attack detection rate.

V. The Automatic Threat Detection

An important characteristic of the proposed system to allow real-time detection is the elimination of intervention from security experts to classify threats and configure the system. This is a major source of errors and an important factor that slows down the threat detection. The proposed system relies on machine learning algorithms to perform automatic threat classification. Thus, the algorithms are responsible to discover the characteristics from each class of attack, instead of requiring manual configuration as in current security systems. Moreover, the feedback of flow data to the off-line processing enables dynamic updates for the system and, as a consequence, the system acquires an adaptive behavior.

In Sections V-A, V-B, and V-C, we present the classification algorithms implemented to evaluate the system. We selected these algorithms because they are among the most used for network security [3]. In all methods, the training is performed with 70% of the dataset and the test with the remaining 30%. During the training phase, we perform ten fold cross validation to avoid overfitting. In cross validation, parts of the dataset are divided and not used in model parameters estimation. They are further used to check whether the model is general enough to adapt to new data, avoiding overfitting to training data.

A. Decision Tree Algorithm

In decision tree, leafs 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. The real-time implementation of the decision tree consists of in if-then-else

rules that generate the tree-like structure previously calculated. The results are presented in the Section V-F, along with the ones from the other algorithms.

B. Artificial Neural Network Algorithm

The artificial neural networks are based on the human brain, in which each neuron performs a small part of the processing, transferring the result to the next neuron. In artificial neural networks, the output represents a degree of membership for each class, and the highest degree is selected. The weight vectors Θ are calculated during the training. These vectors determines the weight of each neuron connection. In the training, there are input and output sample spaces and the errors, caused by each parameter. Errors are minimized through the back propagation algorithm.

In order to determine to which class a sample belongs 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 that leads 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, classifying the sample as the highest one.

C. Support Vector Machine Algorithm

The Support Vector Machine (SVM) is a binary classifier, based on the concept of a decision plane that defines the decision thresholds. Basically, SVM classifies through the construction of a hyper-plane in a multidimensional space that split different classes. An iterative algorithm minimizes an error function, finding the best hyper-plane separation. This hyper-plane is defined by a kernel function. This way, SVM finds the hyper-plane of maximum margin, that is, the with the biggest distance possible to both classes.

The real-time detection is performed by the classification to each one of the classes: normal and non-normal; DoS and non-DoS; and probe and non-probe. Once SVM calculates the output, the chosen class is the one with the highest score. The classifier score of a sample x is the distance from x to the decision boundaries, that goes 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 $(\alpha_1, ..., \alpha_n . b)$ 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 presents a good performance with the minimum quantity of input parameters.

D. Window Size Determination

We evaluate the accuracy of all the three analyzed algorithms for different window sizes. The results are presented in Figure 3. Both neural network and SVM achieved a good detection starting from a window size of one second, while decision tree only achieved similar performance with a two-second window. We choose the one-second window size, because flow composition has enough gathered information to correctly classify the samples with shortest possible time. In the following experiments, only SVM and neural network are used because they present an accuracy above 95%. SVM produces a robust classifier, due to the maximization of the margin between the hyper-plane and the separated samples. On the other hand, neural network presents a good performance, due to its ability to adapt to nonlinear data.

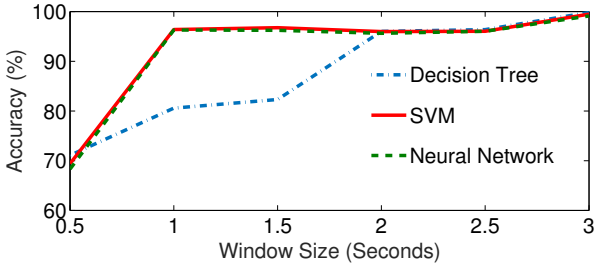


Figure 3: Accuracy of decision tree, SVM, and neural network algorithms for different flow window size. Both SVM and neural network achieved an accuracy above 95% starting from one second flow window.

E. Feature Selection and Principal Component Analysis

To improve the efficiency of the proposed system in real-time threat detection, we perform dimensionality reduction. The aim is to eliminate irrelevant features for the threat detection procedure. Another aspect to be considered is a possible correlation between two or more features, which can be combined into only one feature, reducing the processing time.

The dimensionality reduction is achieved through the Principal Component Analysis (PCA), which transform a group of possibly correlated variables into a group of linear uncorrelated variable that lie in orthogonal planes. This transformation takes into account the eigenvalues in a way that the component associated to the biggest eigenvalue represents greater data variance. The other components of the resulting matrix are also sorted in represented data variance order. Then, we keep the components associated to the higher eigenvalues, because they have more relevant information and we withdraw the components associated to the 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 shows the eigenvalues associated to the elaborated dataset to evaluate the proposed system. The sum of the

higher eight eigenvalues represents 95% of the total sum. In other words, the first eight features from the data calculated by the PCA linear transformation represent 95% of the total variance. Therefore, these eight components are selected and the others, that represents only 5% of the total data variance, are discarded, improving the processing time, which is critical in real-time applications.

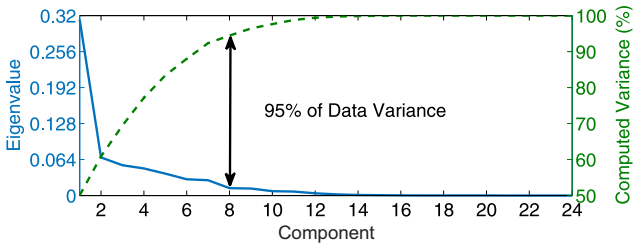


Figure 4: Eigenvalue for each of the 24 flow feature. The eigenvalue associated to each of the transformed features is proportional to the data variance. The eight highest principal components represent 95% of the total data variance.

F. Results of Attack Classification

We show the obtained classification results of the SVM and neural network classifiers through two metrics: accuracy and confusion matrix. The confusion matrix clearly specifies the false positive rate and other metrics of each class in the test dataset. This matrix is in the format real class versus predicted class. The lines represents the elements that belongs to the real class and the columns the elements that were predicted to belong to the class. Therefore, diagonal elements of this matrix represent the number of elements that are correctly classified, since they belong to the predicted class. We choose this metric, because all other metrics, like false positive rate, true positive rate, and others, can be derived from this matrix.

Table I: Accuracy comparison between neural network and SVM for both the original dataset and the one after the PCA feature selection.

	Neural Network	**SVM**
Full Dataset Accuracy	96.2960%	96.3757%
PCA Dataset Accuracy	95.9602%	96.3119%

Table I shows that even after the feature selection using PCA, both algorithms present accuracy above 95%. The PCA dataset is the one obtained after the transformation and selecting the eight first transformed features. Therefore, according Table I, there is less than 1% loss in accuracy when reducing from 24 to eight features through PCA, improving considerably the processing time.

Tables II and III show the confusion matrices of the applied algorithms in attack classification. In the main diagonal, it is observed the correctly classified samples in each class. In this way, the accuracy is analyzed in more details. Both Tables present similar performance, but SVM presents a slightly better result.

Table II: Neural Network Confusion Matrix.

	Normal	**DoS**	**Probe**
Normal	38962	477	1
DoS	341	12116	0
Probe	1777	0	10586

Table III: SVM Confusion Matrix.

	Normal	**DoS**	**Probe**
Normal	38965	474	1
DoS	119	12338	0
Probe	1776	0	10587

G. Anomaly Detection by Normal Distribution

Zero-day attacks are very hard to detect, since there are not yet any data about the attack. Hence, the protection against unknown attacks is essential to have a higher level of security in computer networks. The anomaly detection has the capability to discover new attacks.

In this paper, we propose anomaly detection by the sample feature distance from a normal distribution. Therefore, anomalies are detected through the mean and variance from each feature of the normal samples of the training dataset. This way, anomalies are detected 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. The eight PCA transformed features are analyzed.

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:

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

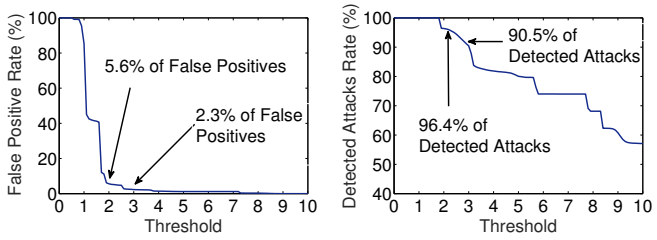
The proposed system allows real-time anomaly training. Consequently, the algorithm becomes adaptive, which is fundamental for anomaly detection, since the network behavior may change in time. Therefore, when a new sample arrives in the system and it is not detected as an anomaly by conditions (5) and (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:

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

Therefore, current values of the sum and the total of samples N are stored in the system and incremented when a new sample arrives, considering each feature X_j. Like this, the normal distribution parameters are always updated by samples considered legitimate, ensuring adaptability.

Figure 5 shows the results for different threshold values. The false positive rate was obtained by the normal test dataset and the attack detection rate by all the attacks in the dataset. When choosing a low threshold value, almost all attacks are detected,

however at the cost of a high false positive rate. On the other hand, a high threshold value results in less false positives, but also in a lower attack detection rate. With a threshold value of two, the false positive rate is 5.6% and the detection rate 96.4%. With a threshold of three, these rates are 2.3% and 90.5%. The threshold parameter must be chosen depending on the application and considering the trade-off between the false positive and attack detection rates.



(a) False Positive Rate. (b) Attack Detection Rate.

Figure 5: a) False positive and b) attack detection rates according to the threshold. The lower the threshold, more attacks are detected, but also higher is the false positive rate.

H. Real-Time Performance

The threat detection time is to guarantee network security. If the detection takes a long time, no reaction can neutralize the threat. Through stream processing and the lambda architecture, it is possible to use parameters from an off-line training for a real-time detection. A prototype of the system was implemented in an environment with four virtual machines running on FITS (Future Internet Testbed with Security) [17] to evaluate the real-time performance. The virtual machines executed on a physical machine with Intel Core i7-2600 processor and 16 GB RAM memory. The detection time is given for the neural network algorithm to provide the worst scenario, because it presents higher computational complexity among implemented algorithms. The system was able to process, in average, 3.57 million samples per minute with variance of 156 thousand, obtained with 95% confidence interval. Therefore, the detection time of each attack is approximately 17 microseconds, which allows defense strategies, decreasing risks.

VI. Conclusion

This paper proposes a real-time stream processing threat detection system. The proposed system uses the lambda architecture, which combines stream and batch processing. The off-line batch processing allows automatic updates of machine learning algorithms to be used in real-time. To evaluate the proposed system performance, a dataset was created from real network traffic. The captured packets were abstracted into labeled flows with 24 features. To increase system efficiency, the dataset dimensionality was reduced by using the principal component analysis algorithm. Three machine learning algorithms, decision tree, neural network and support vector machine, were implemented to perform real-time classification and two of them obtained accuracy above 95%. Moreover, we implemented an anomaly detection algorithm to detect zero-day attacks with a good trade-off between false positive and attack detection rates, when adjusting the threshold. The anomaly detection algorithm is also adaptive, since parameters are updated in real-time. The implemented prototype uses stream processing tools, such as Apache `Storm` e o Apache `Kafka` and, therefore, achieve low threat detection time, enabling defense strategies.

Acknowledgment

This work was supported by CNPq, CAPES, and FAPERJ.

References

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

[2] 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.

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

[4] K. Rai and M. S. Devi, "Intrusion detection systems: A review," *Journal of Network and Information Security*, vol. 1, no. 2, 2013.

[5] M. Andreoni 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.

[6] A. Sperotto, R. Sadre, F. Van Vliet, and A. Pras, "A labeled data set for flow-based intrusion detection," in *IP Operations and Management*. Springer, 2009, pp. 39–50.

[7] 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.

[8] 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.

[9] T. Johnson and L. Lazos, "Network anomaly detection using autonomous system flow aggregates," in *Global Communications Conference (GLOBECOM), 2014 IEEE*. IEEE, 2014, pp. 544–550.

[10] Y. Du, J. Liu, F. Liu, and L. Chen, "A real-time anomalies detection system based on streaming technology," in *Sixth International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, vol. 2. IEEE, 2014, pp. 275–279.

[11] S. Zhao, M. Chandrashekar, Y. Lee, and D. Medhi, "Real-time network anomaly detection system using machine learning," in *11th International Conference on the Design of Reliable Communication Networks (DRCN)*. IEEE, 2015, pp. 267–270.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] I. M. Moraes, D. M. Mattos, L. H. G. Ferraz, M. E. M. Campista, M. G. Rubinstein, L. H. M. Costa, M. D. de Amorim, P. B. Velloso, O. C. M. Duarte, and G. Pujolle, "FITS: A flexible virtual network testbed architecture," *Computer Networks*, vol. 63, pp. 221–237, 2014.