A FAST AND ADAPTIVE THREAT DETECTION AND PREVENTION ARCHITECTURE

Antonio Gonzalez Pastana Lobato

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Otto Carlos Muniz Bandeira Duarte

Rio de Janeiro
Dezembro de 2017

A FAST AND ADAPTIVE THREAT DETECTION AND PREVENTION ARCHITECTURE

Antonio Gonzalez Pastana Lobato

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2017

Lobato, Antonio Gonzalez Pastana

A Fast and Adaptive Threat Detection and Prevention Architecture/Antonio Gonzalez Pastana Lobato. – Rio de Janeiro: UFRJ/COPPE, 2017.

XVI, 64 p.: il.; 29, 7cm.

Orientador: Otto Carlos Muniz Bandeira Duarte

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2017.

Referências Bibliográficas: p. 59 – 64.

1. Machine Learning. 2. Threat Detection and Prevention. 3. Stream Processing. I. Duarte, Otto Carlos Muniz Bandeira. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

À minha família.

Agradecimentos

Gostaria de agradecer principalmente à minha família. Sem o apoio dela não seria possível realizar este trabalho. Um agradecimento especial aos meus pais e avós por todo o tempo e esforço gastos com a minha formação.

Agradeço à UFRJ e a todos os professores, em especial aos professores do Departamento de Eletrônica e do Grupo de Teleinformática e Automação, por toda a participação em minha formação. Em especial, agradeço ao meu orientador, professor Otto Carlos Muniz Bandeira Duarte, por todos os conselhos e dedicação durante a orientação. Uma menção especial de agradecimento ao colega Martin Andreoni Lopez pela ajuda e discussões neste trabalho. Gostaria de agradecer também a todos os colegas do GTA/UFRJ pelo ótimo ambiente de trabalho no qual este trabalho foi desenvolvido.

Esse trabalho foi realizado com apoio do CNPq, CAPES, FAPERJ, and FAPESP (2015/24514-9, 2015/24485-9, and 2014/50937-1).

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

UMA ARQUITETURA PARA DETECÇÃO ADAPTATIVA E RÁPIDA PREVENÇÃO DE AMEAÇAS

Antonio Gonzalez Pastana Lobato

Dezembro/2017

Orientador: Otto Carlos Muniz Bandeira Duarte

Programa: Engenharia Elétrica

A detecção tardia de ameaças aumenta significativamente o risco de danos irreparáveis, desabilitando qualquer tentativa de defesa. Esse trabalho propõe uma Arquitetura rápida e adaptativa para Detecção e Prevenção de Ameaças baseada em processamento de fluxos e algoritmos de aprendizado de máquina. A arquitetura proposta combina a adaptabilidade de algoritmos de aprendizado de máquina treinados em tempo real com a eficiência de métodos treinados em lote. Um conjunto de dados foi criado através da captura tanto de tráfego legítimo, quanto de tráfego malicioso. Dois modos de se combinar os pacotes em fluxo são avaliados: um agregando todos os pacotes em uma janela de tempo; e o outro analisando apenas os primeiros pacotes de um fluxo. Além do conjunto de dados criado, um conjunto de dados contendo tráfego de usuários de uma das maiores operadoras do Brasil também é utilizado. Para avaliar a arquitetura de detecção proposta, cinco algoritmos de classificação e dois de anomalia são desenvolvidos. A arquitetura proposta provê ótima adaptabilidade, detectando novas ameaças em tempo real, e um bom compromisso entre taxa de detecção de ameaças e de falsos positivos na detecção por anomalias. Um esquema baseado em Redes Definidas por Software, que automaticamente previne ameaças apenas com informações dos primeiros pacotes de um fluxo, também é proposto. A proposta eficientemente bloqueia ameaças, é robusta e escala de acordo com a demanda, mesmo em cenários nos quais os atacantes usam IP mascarado. Além disso, a escalabilidade é avaliada, ao aumentar o número de núcleos de processamento de fluxos e alocar mais recursos para elementos sensores. Os resultados mostram uma alta acurácia, acima de 90% e um tempo de detecção de ameaças de quatro microsegundos, o que permite o disparo imediato de contramedidas.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

A FAST AND ADAPTIVE THREAT DETECTION AND PREVENTION ARCHITECTURE

Antonio Gonzalez Pastana Lobato

December/2017

Advisor: Otto Carlos Muniz Bandeira Duarte

Department: Electrical Engineering

The late detection of security threats causes a significant increase in the risk of irreparable damages, disabling any defense attempt. We propose a fast and adaptive Threat Detection and Prevention Architecture based on stream processing and machine learning algorithms. The proposed architecture combines the adaptability of online trained machine learning algorithms with the efficiency of batch trained methods. We create a dataset by capturing both legitimate and malicious traffic and compare two ways of combining packets into flows, one gathering all packets in a time window and the other analyzing only the first few packets of each flow. Besides our created dataset, we also analyze our proposal on real data composed of fixed-broadband Internet user traffic from one of the major Brazilian network operators. In order to evaluate our detection architecture, we develop five classification algorithms and two anomaly detection methods. In fact, the proposed architecture provides adaptability to new traffic behavior and achieves a high accuracy rate and a good trade-off between attack detection and false positive rate in anomaly detection. We further propose an improved scheme, based on Software Defined Networks, that automatically prevents threats by only analyzing the first few packets of a flow. The proposal promptly and efficiently blocks threats, is robust, and can scale up, even on scenarios in which the attacker employs spoofed IP address. Moreover, we evaluate the scalability, by increasing the number of stream processing cores and allocating more resources to sensor elements. The results shows an accuracy higher than 90% and threat detection time of four microseconds, which promptly enables counter measures.

Contents

List of Figures

List of Tables

Acronyms

CPU - Central Processing Unit

DAG - Direct Acyclical Graph

DDoS - Distributed Denial of Service

DoS - Denial of Service

DTW - Dynamic Time Warping

FITS - Future Internet Testbed with Security

FNT - Flexible Neural Trees

GRE - Generic Routing Encapsulation

ICA - Independent Component Analysis

ICMP - Internet Control Message Protocol

IDS - Intrusion Detection System

IoT - Internet of Things

IP - Internet Protocol

IPS - Intrusion Prevention System

KDD - Knowledge-Discovery in Databases

OPF - Optimum-Path Forest

PCA - Principal Component Analysis

R2L - Remote to Local

SDN - Software Defined Networking

SGD - Stochastic Gradient Descent

SIEM - Security Information and Event Management

SOM - Self Organizing Map

SVM - Support Vector Machines

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

Chapter 1

Introduction

The huge amount of data transferred in current communication networks creates a challenging scenario for threat detection [1]. Threats and security attacks are spread and tend to increase significantly in the future with the Internet of Things (IoT), since more than 80 billion devices are estimated to be interconnected by 2025 [2]. This scenario implies a high monitoring and protection complexity, with several challenges in security and data privacy. The billions of devices generate a big amount of data streams, which need to be managed, processed, transferred, and stored in a secure real-time way, avoiding important information loss. Moreover, the big data characteristics of velocity, volume, and variety increase the number of vulnerabilities to be explored by attackers. As a consequence, is mandatory to ensure data security and privacy as well as the security of services and infrastructures, especially considering cloud environments [3].

Attackers keep changing their behavior and searching unknown vulnerabilities to surpass traditional security mechanisms. Therefore, zero-day attacks are increasing by more than 125% each year, and it is expected to appear one new attack per day by the year 2021 [4]. This kind of attack is hard to identify, because threat related information becomes available, on average, 312 days after the attack occurs [5]. In this sense, honeypots complement traditional security mechanisms by performing two main tasks, to confuse attackers in order to protect the infrastructure, and to collect information about attacker behavior and attack patterns [6]. Thereby, honeypots are able to spot zero-day attacks and give insights on attacker actions and behavior.

The threat detection time is of the essence to maintain security in cloud computing systems. If detection takes too long, irreparable damages will occur. The effective threat detection demands monitoring, processing, and management of data, in order to extract useful information from network traffic [7]. Current security systems, such as Security Information and Event Management (SIEM), designed to gather and analyze data in a single point, are not effective, since 85% of network intrusions are detected weeks after they had happened [2]. Therefore, the long threat detection time makes any kind of defense unfeasible. We claim that analytic techniques with real-time stream processing enable the immediate analysis of different kinds of data and, consequently, they empower threat detection.

Threat detection alone is not enough to guarantee security. After the detection, the attacker network traffic, even when attackers spoof their IP addresses, must be effectively blocked, without compromising legitimate users. Threat Prevention Systems have two modes of operation, the on-path and the off-path modes. In the first one, the system analyzes the traffic before forwarding it to the destination. The greatest advantage of this mode is that the system can act directly to block the traffic. However, this approach presents downsides, since it introduces latency and cannot correlate traffic from other sources. On the other hand, the second approach, in which the traffic is duplicate from multiple sources and forwarded to the Threat Prevention System, does not present these downsides, but needs an additional scheme to act on network traffic. Both modes still need to adapt to high rates of incoming traffic, to be able to inspect all packets without missing any threat. The Threat Prevention System should avoid idle resources, scaling up when necessary, and releasing the resources when the traffic rate decreases.

We propose and implement a fast and adaptive Threat Detection and Prevention Architecture. Our Threat Detection Architecture combines several open source platforms. The integrated architecture allows big data analysis in a stream processing manner. Our architecture considers traffic from two sources, the legitimate users, collected by the sensors spread in the network and the honeypots, to gather a real-time feed of new threat data, enabling the architecture to automatically adapt itself to new attacker behavior without any manual intervention. The processing cluster receives the collected traffic data through tunnels that connect the distributed sensors and honeypots to the buffer. The buffer, implemented with the publish/subscribe paradigm, adapts different incoming and processing data rates. The detection core is a stream processor, in which we implement the proposed threat detection algorithms. The stream processor analyzes the traffic data in real time, considering the data as a unbounded stream, processing the data as it arrives. Since our detection architecture aims to detect threats as fast as possible, we analyze the performance of three main open-source stream processing platforms to choose the most suited for the detection based on the processing throughput. The analyzed data is then stored in a database to enable historical analysis using batch processing. This offline analysis can be further used to adapt parameters of the real-time detection, enhancing even more the adaptability of the proposed architecture.

The proposed architecture combines conventional batch processing over a historical database with real-time stream processing analysis. We use both historical data to train offline classification algorithms and real-time traffic data coming from honeypots to train both online threat classification and anomaly detection methods. Batch processing classification methods tend to be more accurate in scenarios where the threats are known and there is no significant change in user behavior, because of the analysis of the whole dataset at once, therefore being able to relate all the data features to the classes. However, these methods do not detect zero-day attacks and have to be trained once again to adapt to new usage patterns. Therefore, we propose a real-time threat detection platform that integrates the use of distributed stream processing with the collected data from honeypots to train online machine-learning algorithms and the use of a historical database to further improve threat detection. We consider all honeypot traffic as malicious, since there are no useful services at the honeypots [8], and use this information to update the online threat detection models. The methods that are trained online have greater capability to adapt to the incoming data streams. We propose the use of both classification and anomaly detection algorithms with real-time training. The classification algorithms are able to detect more accurately the new threats that are performed against the honeypots, whereas the anomaly detection algorithms complement the defense by comparing the user behavior with normal network usage, therefore, detecting anomalous threat behavior in other machines, rather than only the honeypots. Consequently, our platform adapts itself to attacker and legitimate behavior changes and learns zero-day attacks. Our proposed architecture features both online supervised threat classification and unsupervised anomaly detection. Furthermore, we create a dataset with labeled classes for the architecture evaluation, containing normal network usage and several attacks, collecting the data through packet captures. Not only we evaluate our architecture with the created dataset, we also use another dataset with real broadband user data from one of the most important network operator in Brazil. We propose and analyze two ways of combining the packets into flows to extract features for the threat detection. The first one gathering all packets in a time window and the second, only the first packets of each flow, considering a flow as a sequence of packets from the same source IP to the same destination IP within a time window. Therefore, the first few packets of each flow are periodically analyzed. We implement seven detection methods, five supervised classification methods, two with real-time training and three with batch processing over a historical database, to detect known threats and two real-time unsupervised anomaly detection methods to detect zero-day attacks and unknown threats. The results show a high accuracy for known threats, higher than 90% as traffic streams arrive. The implemented anomaly detection methods present an exceptional trade-off between false positive and detection rates, proving that the architecture can model well legitimate user behavior, even when it changes over time.

Our proposed architecture scales up with the increase in network usage and release idle resource to avoid additional costs. With the increase in network usage in the past few years [2] and the usage pattern that tend to vary over time, considering higher usage during some periods of the day, one of the architecture features is to adapt itself to this inconstant traffic rate. We analyze the behavior of the network sensor, to guarantee that they are able to extract the features of network flows. We measure the processing required as the traffic grows. We implement these sensors in virtual machines, therefore we dynamically create new machines and mirror part of the incoming traffic when an overload is detected. Due to the flexibility provided by the virtualization, we can also shutdown sensor machines, redirecting its traffic to another virtual sensor machine with idle resources. Therefore, we allocate only necessary resources to the sensor elements, reducing costs and optimizing physical resource usage. Regarding the stream processing core of the architecture, responsible to run the proposed methods on the data sent by the sensor methods, we can scale up and down by changing the parallelism of our threat detection application. This is achieved, thanks to the distributed stream processor of our architecture. We achieved a threat detection time of four microseconds when scaling up the parallelism of the threat detection. Thus, our architecture as a whole has an elastic behavior, adapting to different traffic and processing rates.

We further propose a scheme based on Software Defined Networks [9] and the analysis of the first few packets of a flow to implement Threat Prevention, with a strategy to block threats with spoofed IP addresses. The scheme mirrors only five packets to the sensor elements per period and effectively blocks threats as near as possible to its origin based on the source IP. This scheme is robust and easily scalable, since it protects the sensor elements against flood attacks, by sending only few packets to be analyzed. However, a vulnerability would be the use of spoofed IP addresses by the attackers that would overload the controller, with the huge amount of rules to be created. To address this issue, we propose a strategy against spoofed IP threats, based on the time between alerts. When two alerts arrive within a short time, it indicates that the attacker changed its IP and surpassed the blocking rule. Therefore, we monitor the flows to discover through which switch and port the attacker sends the attack and when a third alert arrives, the controller knows exactly from where the attack comes and can perform a counter-measure. The controller only monitors the flows in the case of spoof IP threat is detected and leaves this mode after the monitoring period to avoid the waste of resources. Hence, our scheme is robust, since it protects both the sensors against flood attacks and the controller against spoofed IP threats.

1.1 Contributions and Publications

The proposed Threat Detection and Prevention Architecture as a whole presents several advantages, such as:

1. Effective and adaptive threat detection for both known and unknown threats. In the proposed architecture, we train adaptive algorithms in real time, learning zero-day threats, together with batch-trained algorithms.

2. Fast counter measure 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.

3. Robustness against flood attacks and with a high potential for scalability, since the analysis machine only receives few packets per flow.

4. Elastic behavior since it scales up and down both the number of processing cores and dynamically allocate sensor elements according to the demand.

5. Effective in the threat 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.

As a direct result from the contributions of this work, the following papers were elaborated:

- "Uma Arquitetura Elástica para Prevenção de Intrusão em Redes Virtuais usando Redes Definidas por Software", published in XXXII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2014 [10]

- "Um Sistema Acurado de Detecção de Ameaças em Tempo Real por Processamento de Fluxos", published in XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2016 [11]

- "Um Sistema Adaptativo de Detecção e Reação a Ameaças", published in XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg'17 [12]

- "An Adaptive Real-Time Architecture for Zero-Day Threat Detection", to appear in IEEE International Conference on Communications - ICC'2018 [13]

- "A Fast and Accurate Threat Detection and Prevention Architecture using Stream Processing", to be submitted to Computer Communications [14]

Moreover, as a co-author and indirect result, this work collaborated to the elaboration of the following papers:

- "A Performance Comparison of Open-Source Stream Processing Platforms", published in IEEE Global Communications Conference - GLOBECOM'2016 [15]
- "Um Algoritmo Não Supervisionado e Rápido para Seleção de Características em Classificação de Tráfego", published in XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2017 [16]
- "Collecting and Characterizing a Real Broadband Access Network Traffic Dataset", published in 1st Cyber Security in Networking Conference - CSNet'17 [17]

1.2 Organization

The rest of the work is organized as follows. Chapter 2 discuss related work. The proposed Threat Detection Architecture is presented in Chapter 3. Both datasets to evaluate the architecture are presented in Chapter 4. In Chapter 5 the dataset parameters determination, the threat detection methods and results are discussed. Chapter 6 presents the proposed scheme for threat prevention. Finally, Chapter 7 concludes the work.

Chapter 2

Related Work

Machine learning techniques have been successfully applied to detect threats in computer networks. These techniques can be either supervised or unsupervised, depending on whether the dataset is labeled or not, respectively. The supervised analysis classifies attacks. Among the most well-known classification techniques are neural networks, decision trees, and Support Vector Machines (SVM) [18]. Li *et al.* combine a pattern matching technique, Dynamic Time Warping (DTW), and SVM to generate intrusion detection rules [19]. The method is evaluated with the traditional KDD dataset, to classify between Denial of Service (DoS), Probe, and Remote to Local (R2L) attacks. However, their method can only deal with known attacks. In the unsupervised analysis, there is no information regarding the class to which each sample belongs. Pattern detection applies this kind of analysis. 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 [20]. Common methods to detect outliers apply the 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, in order to perform profile-based for anomaly detection [21]. Palmieri *et al.* use Independent Component Analysis (ICA) to separate the network traffic from their different sources [22]. Then, decision trees are applied as to perform a binary classification. However, PCA and ICA are both sensitive to noise when used in anomaly detection [23]. An improved approach using Tsallis entropy in order to detect anomalous traffic is proposed by Amaral *et. al* [24]. This technique is applied to real and simulated traffic, but it only considers six features for the anomaly detection. In our proposal, we improve the sample entropy metric by employing a time series that takes into account 26 features of the traffic. Moreover, we implement six other algorithms for threat detection to complement this method.

Bostani and Sheikan propose an unsupervised anomaly detection algorithm using MapReduce [25]. The work uses Optimum-Path Forest (OPF) algorithm to project clustering models and detect anomalous behaviors. Nevertheless, the paper only focuses on two specific attacks, sinkhole and selective-forwarding, disregarding unknown threats or zero-day attacks. A similar platform is proposed by Singh *et. al.*. The authors use MapReduce most popular implementation, Hadoop, to perform big data analytics. In this case, big data analytics are used to detect Peer-to-Peer botnets [26]. A limitation of this approach is that the entire analysis is executed offline. Rathore *et al.* [27] proposed a Hadoop based real-time Intrusion Detection System (IDS) for detecting threats in high-speed networks. Unlike our proposal that also trains the model and classifies threats in real-time, they use the term real-time referring to stream processing classification with offline training. Moreover, Rathore *et al.* proposal and many others conventional traffic classification research are based on artificial and outdated datasets, such as the KDD99 [28] dataset. In this paper, we use two real traffic datasets, with packet captures from our lab and one of the major network operators in Brazil.

The growing number of applications and users in the Internet motivated the development and deployment of honeypots [29]. Honeypots are used to discover and gather data about new threats. In honeypots, applications are deployed without any useful data to trick attackers. Owerzarski *et al.* [30] proposes an unsupervised clustering algorithm for anomaly characterization of honeypot traffic. Their approach assumes that all honeypot traffic is malicious and classifies it into different anomalies. In contrast, our work performs both supervised and unsupervised detection, detecting threats among legitimate network traffic. Our work is able to use honeypot data to train classification methods, therefore adapting itself to new threats.

Bernaille *et al.* [31] propose the use of the first packets of each flow to determine the application of the analyzed traffic. They claim that applications presents a well-defined behavior at the beginning and, therefore, the analysis of the first packets is enough to classify applications. The work proposes a scheme to classify applications that is heavily based on the packet size feature. The authors state that their work is vulnerable to attackers who can adjust this parameter to trick the system. Our work analyzes 26 features of each flow, therefore making it unfeasible for attackers to adjust all parameters to trick the system. Peng *et al.* also utilize the first packets applied to eleven supervised machine learning algorithms to identify Internet traffic [32]. The first few packets for traffic identification approach is also applied by Chen *et al.* [33]. Flexible Neural Trees (FNT) are applied to classify network applications. However, the work focuses on traffic characterization, not detecting threats, and process data offline. Donato *et al.* propose an open source platform for application traffic identification [34]. Our proposed platform uses six machine learning algorithms and achieve a good traffic identification.

Another important aspect approached by researches is the creation of datasets to analyze, evaluate, and compare different threat detection systems. Sperotto *et al.* [35] created a dataset with over 14 million of labeled flows. A number of other works also create their own dataset to evaluate their proposals [36, 37]. 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 [38] propose anomaly detection by aggregating IP flows. The results, however, are not analyzed for real-time scenarios. Du *et al.* [39] employ a stream processing platform to detect anomalies. Nevertheless, they only analyze incoming packet rates and therefore only detect peaks that not necessarily correspond to anomalies, as in the case of flash-crowds, obtaining possibly a high false positive rate. Zhao *et al.* [40] also use a stream processing platform to detect network threats, but their results are preliminary.

Snort and Bro are the most popular open source real-time intrusion detection tools [41]. 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, in which the user must create its own applications. In contrast with our work, these traditional IDS do not adapt to new threats, since Snort needs rules related to the new attack and Bro, used as threat detection platform, needs to be programmed to detect this new threat. In our work, we use Bro framework to extract flow features. The threat detection is performed using stream processing and real-time machine learning algorithms that can adapt to unknown threats.

The programmability of Software Defined Networking (SDN) enables the development of security frameworks. Shin *et al.* propose Fresco [42], a modular high level language to develop security applications. The language Pyretic [43] uses SDN and its global view to define states in the network and is used to define which application to run in each state. These frameworks show the potential of SDN to perform security actions, such as mirror, block and deviate the traffic. In our work, we use SDN to mirror the traffic to sensor elements and effectively block threats as near as possible to its source, even in spoofed IP scenarios, due to SDN global view.

Several proposals also use SDN to develop security applications. In previous work, we proposed a combination of Bro IDS and OpenFlow Pox controller [44]. We implemented an algorithm to block a Denial of Service (DoS) attack in Bro that sends messages to the controller in order to automatically mitigate the attack. However, 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 [45]. The work detects DoS attacks and HTTP malicious request. Nevertheless, SDN features are mainly used for load balancing and automatic attack detection is not addressed. Braga*et al.* propose a method to detect Distributed Denial of Service (DDoS) using features collected by a NOX-OpenFlow application [46]. The features are collected from switch flow tables and are used in a Self Organizing Map (SOM) detection method. However, since the controller is responsible for the training of the detection method, it can cause delays and overload in the controller.

Unlike the previously cited papers, we propose a specific architecture that allows real-time stream processing analysis based on the support of a historical database and incoming honeypot data. Hence, our architecture learns and adapts to new attacks captured by the honeypots and monitors legitimate user behavior to detect anomalies. Moreover, we implement our detection methods using stream processing, to detect threats in real-time and to scale using distributed processing. Our architecture prevents threats in a fast and scalable way by providing real-time accurate detection of known and zero-day attacks through automatized classification and anomaly detection methods. In this work, we implement five classification methods, two of them with real-time training, adapting without manual intervention to new threats. In addition to these methods, we also propose two anomaly detection algorithms that are also trained in real-time. Therefore, our work presents a solid threat detect, since it handles known attacks, learns new attacks through honeypot data, and also monitors normal usage behavior to detect anomalies that are potential threats. Moreover, to efficiently protect the network, our innovative proposal benefits from the fast stream processing, the rapid machine-learning analysis of the first few packets of each flow, and, due to the software defined network features, the prompt threat block even when the attackers changes its IP. We use SDN to mirror the traffic to the sensor elements, therefore avoiding any delay in legitimate user communication. Our threat detection architecture sends alerts to the controller, which is able to block the source IP in all network switches. If the attacker changes its IP, this rule is ineffective. Therefore, we propose a scheme to protect the network against these attacks, based on the time between the alerts. The intuition behind this method is that, when two alerts arrive in short period, it represents the possibility of a spoofed IP threat.

Besides scaling the processing capability of the stream processing core of our threat detection architecture, we also evaluate the resources necessary for the sensor elements to extract flow features. We instantiate new sensors when a overload is detected and also disable sensors, when idle resources are detected. Using SDN we mirror the traffic to the sensor elements and when a new sensor is instantiated, we divide the flows between the sensors. On the other scenario, when a sensor is disabled to spare resources, we first deviate the traffic from this sensor to another with idle resources to ensure security during the whole process. To the best of our knowledge, our work is the only one that combines a wide range of known and unknown threat detection methods, adapting to zero-days threats, with threat prevention techniques, and evaluates the performance of the architecture, scaling according the demand, with an elastic behavior.

Chapter 3

The Proposed Threat Detection Architecture

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 [47]. Unlike batch processing, stream processing techniques analyze massive unbounded data that are continuously generated. We propose an architecture that detects threats using stream processing. We implement both online trained and batch trained algorithms to achieve threat detection. Threat detection is a critical application and requires real time responses to enable threat prevention and avoid irreparable damages. In our proposal, we use honeypot data, along with network traffic, to train online methods and detect network threats in real-time. Along with this scheme, we also implement batch methods trained offline based on the lambda architecture that enables big data analysis in a real-time manner [48]. The algorithms are trained offline and the parameters are loaded to the classifiers implemented using stream processing. Therefore, our architecture achieves accurate threat classification for known threats using batch trained algorithms and adaptive detection with the online trained methods that are able to learn new threats based on honeypot data.

The proposed platform architecture uses batch and stream process paradigm, but introduces a new feature in the real-time stream processing to accept and process the data from the Honeypots. The attacks obtained by the Honeypots are promptly used to train online algorithms to detect zero-day attacks, as shown the Figure 3.1. Our architecture allows real-time management and analysis of massive amounts of information, it is divided in four modules: the data capture module, the processing module, the visualization module, and the prevention module. The processing module deals with the incoming data in real-time, training both online and batch trained algorithms and detecting threats. The visualization module analyzes a huge amount

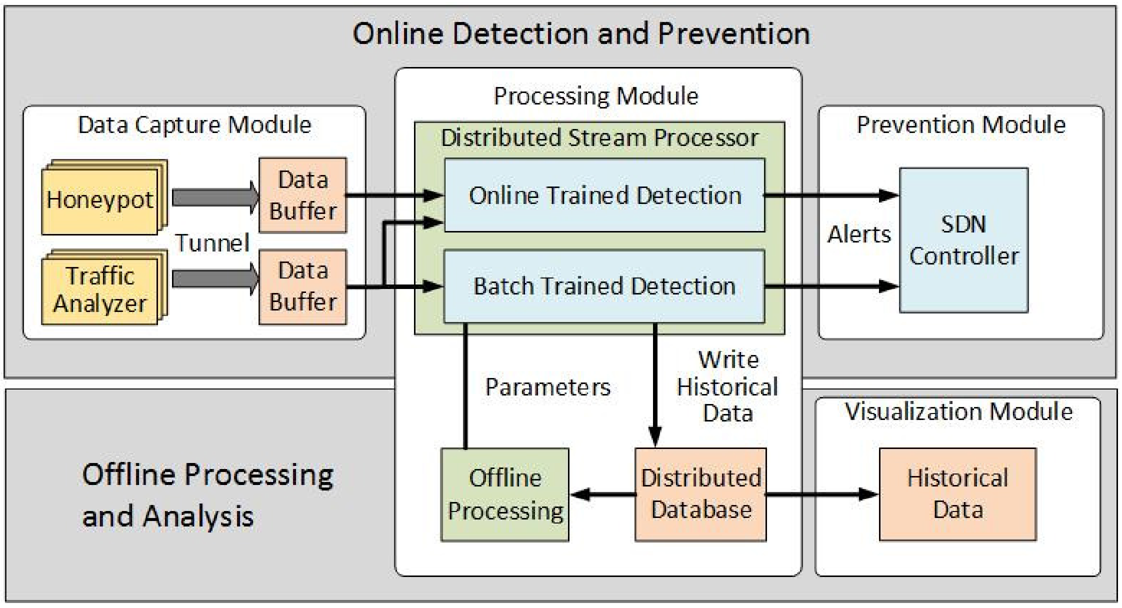


Figure 3.1: The proposed architecture for real-time threat detection is composed of the following modules: i) the Data Capture Module gathers data; ii) the Processing Module analyzes incoming data and detects threats; iii) the Visualization Module displays analytic information results; and iv) the Prevention Module performs counter measures against the threats.

of stored data in a distributed way through techniques such as MapReduce. In our case, the offline processing improves the classification mathematical models. The visualization module combines the obtained information of the previous modules to provide an output composed by analytic data to the user. Finally, the prevention module receives the alerts and performs counter measures against the attacks. The Prevention Architecture is based on Software Defined Networks (SDN) and further explained in Chapter 6.

The data capture module input is heterogeneous, coming from several traffic analyzers and distributed honeypots. In order to capture data from different sources, we deploy the traffic sensors at different network locations using `Bro` framework. `Bro` is a real-time traffic analysis framework that provides its own network oriented programming language, making flow abstraction easier to handle. Therefore, `Bro` characterizes the network flows, synthesizing the packets information and grouping then in time windows. The second type of data source comes from several honeypots. In this work, we use the low-interaction open-source honeypot `Dionaea` to emulate decoy services and log its interaction with the attacker.

The Information is received and abstracted into logs that are transported for further analysis. We detect threats in traffic analyzer data and use honeypot data to adapt our detection schemes in real-time. Apache `Flume` tool acts as a tunnel connecting the data sources to the platform processing core. To avoid data overload, we place a buffer, implemented using Apache `Kafka`, that adapts different generation and processing rates and uses the publish/subscribe model. `Kafka` is a message broker that works as a publish/subscribe service and therefore adapts different production and consumption rates, acting as a buffer. Producers then write their data into topics from which the consumers can access the information.

Apache `Storm` is the stream processor of the threat detection architecture. We choose this platform based on a performance evaluation, presented in Section 3.3, against two other streaming platforms. `Storm` offers a distributed fault tolerant stream processing framework. In addition, `Storm` processes 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, 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 the architecture extracts analytic data from the streams, the results are stored in a dynamic database to a posterior analysis. We use a distributed database to achieve better resilience. The proposed architecture database is the Apache `HBase` that stores massive amounts of spread data and provides real-time access. The stored data contains the information gathered during the threat detection and can be processed offline to calculate parameters to be used in the real-time model. In order to make the architecture more accurate, there is a feedback, since the parameters, calculated offline with historical data, adjust 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.

The Processing Module combines both real-time stream processing with historical data batch processing. The output of this module is then sent to the visualization module, where statistical data about the detected threats are shown. The detection schemes implemented in the processing module operate both based in historical data, similarly to the lambda architecture, or adapting the methods in real time based on the incoming traffic source.

3.1 Real-Time Training Architecture

In our architecture we propose the use of online trained algorithms to adapt to changes in real time. New attacks are detected by the honeypots and sent to the online stream processor. Nevertheless, online classification methods requires labeled data to be updated. We propose to label all data coming from honeypots as threats, since there are not any real service in them and all access are meant to exploit a vulnerability intentionally installed in the honeypots. Therefore, all flows that arrive in the honeypots are labeled as threat and will be used by the algorithms to update the parameters. This real-time data feed ensures an adaptive behavior to the threat detection platform. Consequently, whenever an attacker performs a new attack or changes his behavior, the classification models update.

The architecture considers as normal data the incoming flows from the traffic analyzers in the network during the setup time, which is a period that the network usage is monitored to guarantee that all the flows are legitimate. After this time, when a flow coming from the network analyzer is classified as threat, the platform sends an alert to the prevention module and does not update the parameters. If the flow is considered normal, the platform updates the algorithm parameters, adapting to network behavior changes. This proposed data collection scheme ensures adaptability for both legitimate and malicious behavior. The platform adapts to acceptable changes in the network usage and learns new attacks that are performed against the network honeypots. Zero-day attacks would pass unnoticed by signature based Intrusion Detection Systems or offline classification schemes, however, the proposed data collection scheme is capable to learn these attacks and detect the threat.

3.2 Lambda Architecture and Historical Training

Our proposed architecture is an evolution of the lambda architecture [48], shown in Figure 3.2, to perform real-time threat detection with models trained using batch processing. The lambda architecture allows real-time manipulation and analysis of massive amounts of information and 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 by analytic data to the user. Therefore, the lambda architecture goal is to analyze streaming data accurately, even with its ever-changing incoming rate to obtain real-time results based on historical data. In the lambda architecture the data analysis is divided in three steps: capture, normalization and processing. First, the architecture captures data. Then, every acquired information is sent to the normalization process, in which the features are extracted at high processing rates with external parameters, such as flow window size, number of packets to be analyzed and so on. Hence, the information about the traffic are of higher quality, since the abstraction into flows allows the gathering of more useful information.

The algorithms are trained in the batch processing layer based only in historical data. The algorithms parameters are determined during this training phase and

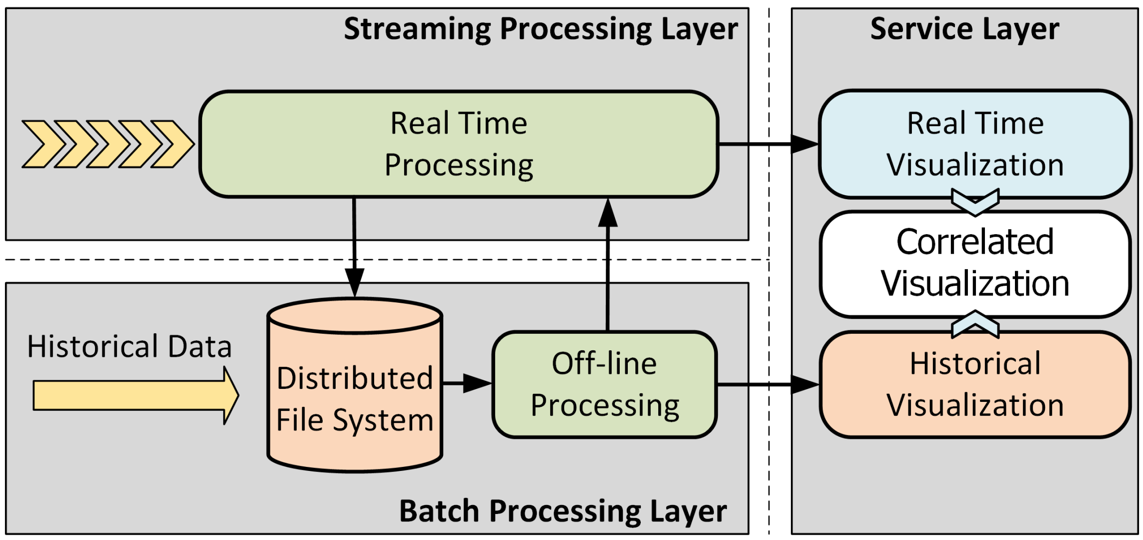


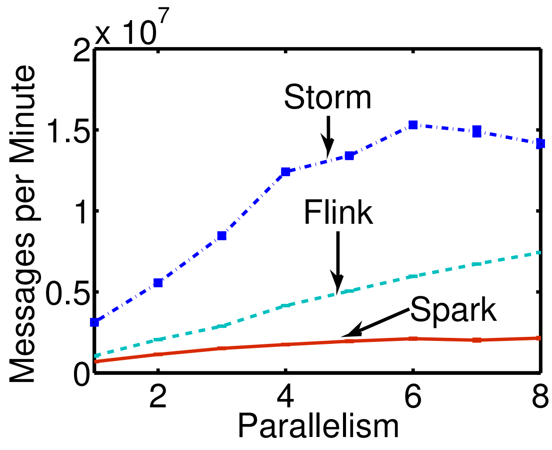
Figure 3.2: The three-layered lambda architecture, which combines stream with batch processing: stream processing, batch processing, and service layers.

then loaded on the stream processing layer. This approach achieves better real time throughput, since there is no additional overload to train the algorithms online [11]. However, it needs to be retrained to learn new threats or adapt to different legitimate behavior. The lambda architecture owns a feedback to periodically update batch processing algorithms, based on stored traffic.

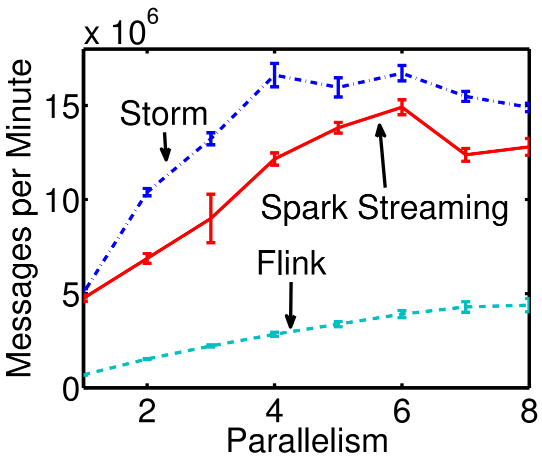
3.3 Stream Processing Performance Evaluation

One of our main concerns is to detect threats as fast as possible. If the detection takes too much time, no reaction can neutralize the threat. For this reason, we evaluate three open-source streaming platforms with our threat detection application to choose the most suited one. We compare [15] two native distributed real time stream processing systems, the `Apache Storm` and the `Apache Flink`, and one micro-batch processing system, the `Apache Spark Streaming`. The native stream processing approach processes each sample as they arrive, whereas the micro-batch gathers the samples and groups then in micro-batches to be processed in a model similar to map reduce. We have evaluated the opensource platforms [15] and the results show that native stream processing systems presents a better throughput, while micro-batch systems are able to recover from failure without any losses. When processing each sample at a time, tracking whether each sample was processed or not is very costly. Therefore, these systems present lower fault tolerance when compared to the micro-batch approach. The sample grouping overhead added in the micro-batch systems affects performance, however it enables to do batch acknowledgement, therefore guarantying that each sample is processed exactly once.

As presented in the previous section, we choose `Storm` to be the processing core of the proposed Threat Detection Architecture based on the processing throughput. We compared `Storm` against `Spark Streaming` and `Flink`. We use our threat detection application as a benchmark to measure the stream processors performance. The detection methods in our application are further presented in Section 5. 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 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 3.3a shows the performance results of this experiment for threat detection. `Storm` shows a much higher throughput than the others do. Figure 3.3a also shows that `Storm` is able to process 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.



(a) Evaluation of the threat detection performance.



(b) Evaluation of the wordcount performance.

Figure 3.3: Throughput results of the platforms in terms of number of messages processed per minute in function of the task parallelism.

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 Million tweets [49]. All three platforms offer the word count application as example. Therefore, we show this result to get an unbiased comparison that is not affected by our implementation. Figure 3.3b shows the performance behavior of the three system under a word count program. Once again `Storm` has a better performance and, therefore, is the most adequate platform for our Threat Detection Architecture.

3.4 The Processing Schemes

The proposed threat detection architecture combines multiple data sources and performs real time threat detection based both on lambda architecture and on real time adaptive training. Figure 3.4 shows the processing schemes that enable threat detection combining real-time stream processing with offline batch processing. Several sensors are spread across the network and monitors the traffic. The network traffic features are extracted and are sent to both historical based and adaptive threat detection methods, implemented in the stream processing core of the platform. Moreover, the architecture stores this information to enable reprocessing of the batch training to update the historical based threat detection algorithms parameters. Another important factor is the historical visualization that enables a performance evaluation of the detection methods. It is crucial to see metrics such as accuracy, threat detection rate, false positive rates, among others, to evaluate the detection models. Based on this visualization, algorithms parameters are tuned and the most suited models are chosen.

Online trained threat detection methods adapts in real time to legitimate behavior changes and are able to learn new attacks as they are performed. When using offline batch processing training, the algorithms are only able to adapt when they are retrained, while adaptive methods incrementally adjust their parameters. To perform this incremental training, several honeypots send their gathered data to perform stream processing training. The data is labeled as threat and is combined with legitimate user traffic labeled by the adaptive threat detection after the setup time, where the training is closely monitored to initialize the detection parameters. Having both legitimate and threat classes labeled in the income stream, the online training updates the classification models incrementally as each sample arrives. Moreover, the adaptive threat detection also performs anomaly detection based only on legitimate traffic. The proposed anomaly detection algorithms model user behavior and mark as an anomaly behaviors that are not within an acceptable threshold.

Both detection approaches send alerts to the prevention schema that immediately triggers counter measures. The historical based detection focuses on detecting known attacks, similar to signature based traditional detection approaches. However, there is a major difference since the machine learning methods requires less human intervention and are easily updated by retraining the methods while signature based methods require that new rules are created and constant maintenance. The online trained detection focus on detecting zero days threats that are the most difficult to be detected, since there are no available information about the attacks and how to discover them. The architecture address this problem using two ap-

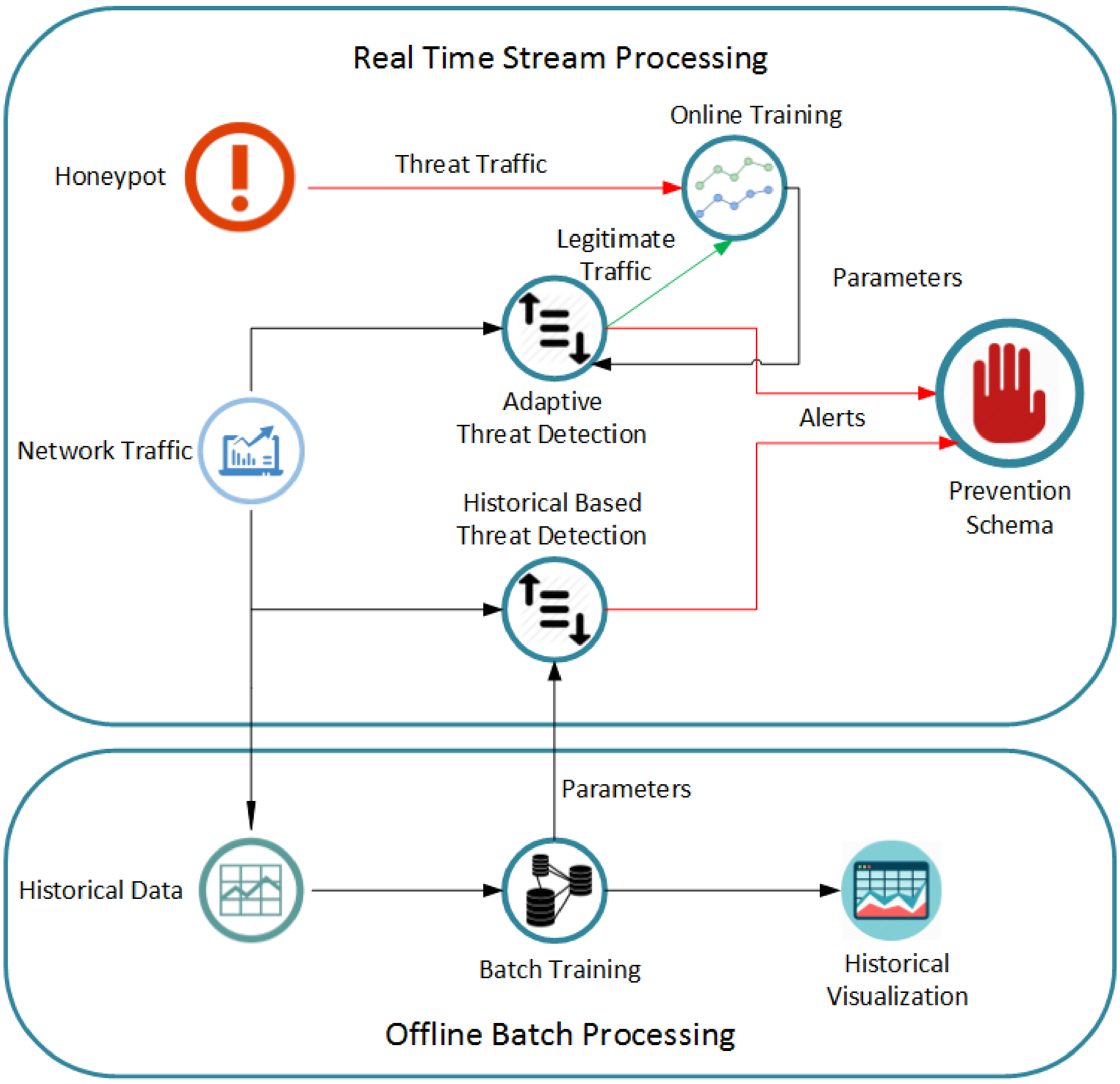


Figure 3.4: Processing Schemes combining stream processing trained adaptive algorithms with offline trained batch algorithms to perform real time detection.

proaches, the first by using a real time feed of honeypot data to train classification methods and the second by using network traffic to model legitimate user behavior and detect anomalies.

Chapter 4

Security Datasets

Only a few network security datasets are available to evaluate defense mechanisms. The main reason to make security data unavailable is due to privacy concerns, since traffic data may contain confidential information. The two best known available datasets are DARPA [50] and KDD 99 [28]. 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 [51]. The most important is that the traffic does not correspond to a real network scenario, since it was simulated. Besides that, there is redundant data, which affects the algorithms results. Another issue is that these datasets are over 15 years old and do not represent current attack scenario [52]. In our work, we evaluate our threat detection methods based on two datasets, one composed with normal traffic from one of the major network operators in Brazil combined with botnets attacks and the other created in our lab GTA/UFRJ. For both datasets, we gather packets into flows. A flow is defined as a sequence of packets from the same IP source to the same IP destination during a time window.

In this work, we evaluate our threat detection proposal using two different datasets, both containing real traffic. One dataset contains real traffic from a Brazilian Network Operator and the other contains real traffic from our lab. The use of two different datasets show that the proposed architecture and its detection methods works well even when considering distinct scenarios. Moreover, we compare 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 time window. 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, shown in Table 4.1, generated by the TCP/IP header data. The main features are: TCP, UDP, and ICMP

Table 4.1: Dataset Features

Number	**Feature**
1	Number of TCP packets
2	Number of source ports
3	Number of destination ports
4	Number of FIN flags
5	Number of SYN flags
6	Number of PSH flags
7	Number of ACK flags
8	Number of URG flags
9	Number of UDP packets
10	Number of ICMP packets
11	Number of IP packets
12	Number of IP types of service
13	Average TTL
14	Average header length
15	Average packet length
16	Number of Do Not Frag flags
17	Number of More Frag flags
18	Average fragment offset
19	Number of RST flags
20	Number of ECE flags
21	Number of CWR flags
22	Average offset
23	Number of ICMP types
24	Number of ICMP codes
25	Mean inter packet arrival time
26	Variance of inter packet arrival time

packet rates; number of source and destination ports; number of each TCP flag; average and variance of inter-packet arriving time; average and variance of flow packet length; among others. The second approach to define flows consists in extract the features from the first few packets of each flow, being periodically analyzed since the flow is defined also by a time window. The intuition behind this approach is that for most applications, the initial behavior is well defined, which leads to a good flow classification. Once again, we determine the number of packets to be considered based on the accuracy of the classification methods.

4.1 Network Operator Dataset

The Network Operator (NetOp) dataset is composed by real-access information from 373 residential broadband users from the city of Rio de Janeiro for a period of one week [17]. Network traffic is anonymized due to privacy concerns. An Intrusion

Detection System (IDS) filtered the traffic. We analyzed the logs from this IDS and the proportion of attacks 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* [53] so that we can detect these threats in order to evaluate our threat detection platform. In the combined dataset, we keep 15% threat traffic proportion. 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 actual malwares.

4.2 GTA/UFRJ Dataset

A contribution of this work is the creation of a dataset with real network traffic to evaluate the proposal[1]. 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 dataset both normal traffic and real network threats, which are composed by seven types of DoS and nine types of probe. The analysis of packet header information is able to detect two threat classes: Denial of Service (DoS) attacks and probe. Therefore, we elaborate 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 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 to the attack machines from the normal lab network usage. The whole dataset has around 95 GByte of packet capture raw data.

4.2.1 GTA/UFRJ Dataset Attack Description

In the Local Area Network Denial of Service (LAND) attack, the attacker sends a TCP packet with a fake IP address SYN tag, containing the victim address and port as the origin and destination. Thus, the system responds with a SYN-ACK packet for itself, creating an empty connection that remains open until the timer ends. Therefore, the system enters into a loop sending responses to itself until eventually collapses. Nestea is an attack that affects Linux servers and the attack teardrop is a similar version for Windows servers. The attack consists of exploiting a vulnerability in the TCP/IP protocol when sending fragmented packets to the victim. When the displacement sum and size of a fragmented packet differ from the next fragmented

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

packet, packets overlap and the victim attempts to mount the packet entering into a denial of service.

Flooding attacks send an abnormal amount of packets without waiting for responses. Therefore, when attempting to process all packets, the victim is unable to respond to all requests on time and causes denial of service. In the proposed data set, there are SYN flood, ICMP flood, and UDP flood threats. In SYN flood, multiple packets with fake source addresses are sent to maintain open connections and thus burst the victim resources. In the ICMP flood, multiple ICMP packets are also sent without waiting for the victim to respond. UDP flood has a different philosophy than previous ones, since UDP is not oriented towards connection. In this attack, packets are sent with the same address, but with different ports, in order to saturate the bandwidth. The Smurf attack, similar to ICMP flood, uses an Echo Request (ICMP) packet with the network broadcast address as the destination address, and victim address as source. The responses to these packets are for the victim, so all the stations of the network respond to the victim. Smurf is considered a distributed denial-of-service attack.

Port scanning consists of verifying which services are active on a server, that is, if there is a TCP port listening to requests and a running process to handle the requests arriving at that port. However, port scanning is considered a threat since most attacks are preceded by a port scan to identify system vulnerabilities that can be exploited. In port scanning, the attacker generates packets and monitors the responses to determine whether a service is vulnerable or not. The SYN scan is best known for simplicity and speed, and is also known as half-open scanning, since it never opens a complete connection. A SYN-packet is sent, and if the system is listening for TCP connections on the port to which the packet is sent, the host will respond with a SYN-ACK packet and then the attacker will send an RST, Closing the connection before the handshake is established. If the service is not open, the victim will respond with an RST packet. If there is no response, after several attempts, the gate is marked as closed, i.e., there is a firewall that prevents communication between the attacker and the victim in such port. The TCP connection scan, also known as full-open scanning, behaves similarly to the previous one, but in this case the communication is complete if the port is open, that is, the attacker sends the SYN, wait for the SYN-ACK, and finally sends the ACK to establish the connection. If the door is closed, the attacker receives an RST/ACK packet.

The TCP FIN, XMAS, ACK, NULL and Maimon attacks are similar, since the absence of response means that the victim port is open or that there is a firewall in the path between the attacker and the victim. In FIN, it sends a TCP packet with the FIN flag active. Depending on the operating system of the victim, if the victim door is open, it will not have the response and, if it is closed, the victim will respond with an RST/ACK packet. TCP XMAS simulates the normal behavior of a client, since it sends TCP packets with the URG, PUSH and FIN flags active in the packet. Again, if the door is open, there will be no response and, if it is closed, you will have an RST/ACK packet response. In NULL scan, unlike FIN scan, a packet with no active flag is sent, and it receives no response when the port is open and a RST/ACK flag when it is closed. The TCP Maimon scan, name received after its creator, sends a TCP packet with active FIN and ACK flags. TCP Window Scan uses different window sizes to an RST response. Depending on the operating system, the open ports use a positive window size while the closed ports have a window of size zero. As an alternative to conventional TCP, UDP and ICMP transport protocols, the SCTP INIT scan was created. This technique has a behavior similar to TCP SYN scanning, which creates a "half" connections. The attacker sends an INIT chunk. An INIT-ACK chunk indicates that the door is open, while an ABORT chunk is an indicative of a closed port. If no response is received after a few retransmissions, the port is marked as closed.

Chapter 5

The Automatic Threat Detection

Our proposed real-time architecture relies on machine learning algorithms and performs automatic threat classification. The automation is an important characteristic, because it reduces the necessity of security expert intervention to classify threats and configure the system. Human intervention is a major source of errors and an important factor that slows down threat detection. Thus, the algorithms are responsible to discover the characteristics from each class of attack, instead of requiring manual signature configuration as in current security systems.

We implemented two classes of threat detection: one based on historical data; and the other with online training using a live feed of honeypot captured data and network traffic. Batch historical training tend to be more accurate, but does not deal well with unknown threats. To handle this problem, we implemented a feedback of flow data to the offline processing that enables dynamic updates for the architecture. This mitigates the problem, but still has a great delay in learning new attacks. On the other hand, online trained adaptive methods learn new threats in real time and profiles normal user behavior to detect anomalies.

5.1 Historical based Threat Detection

In Sections 5.1.1, 5.1.2, and 5.1.3, we present the batch classification algorithms implemented to evaluate the architecture. We selected these algorithms because they are among the most used for network security [18]. 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 tenfold cross validation to avoid overfit. In cross validation, the dataset is divided and a certain number of parts are not used for parameter estimation. These parts are further used to check whether the model is general enough to adapt to new data, avoiding parameter overfit to the training data.

5.1.1 The Decision Tree Algorithm

In decision tree, 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 gain of each feature. The real-time implementation of the decision tree consists in if-then-else rules that generate the tree-like structure previously calculated. The results are presented in the Section 5.1.6, along with the other algorithms results.

5.1.2 The 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 output to the next neuron, achieving complex results from the combination of these small tasks. In artificial neural networks for classification, the final output represents a degree of membership for each class and the output class is determined by highest membership degree. The weight vectors Θ are calculated during the training. These vectors determine the weight of each neuron connection. In the training, the input vectors are mapped onto a predicted output vector that is compared to the real output. The prediction errors are then minimized by the Back-Propagation algorithm, taking into account the error induced by each parameter.

After the training in completed, in order to determine to which class a sample belongs, each neural network layer computes the following equations:

$$z_{(i+1)} = \Theta_{(i)} a_{(i)} \quad (5.1) \qquad a_{(i+1)} = g(z_{(i+1)}) \quad (5.2) \qquad g(z) = \frac{1}{1+e^{-z}} \quad (5.3)$$

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 degree.

5.1.3 The Support Vector Machine Algorithm

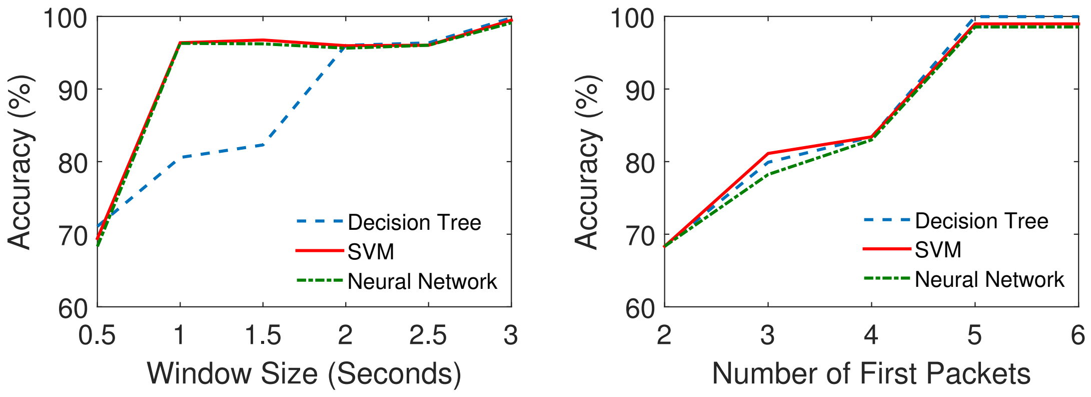
The Support Vector Machine (SVM) is a binary classifier, based on the concept of a decision plane that defines the decision thresholds. Basically, SVM classifies through the construction of a hyper-plane in a multidimensional space that split different classes. An iterative algorithm minimizes an error function, finding the best hyper-plane separation. 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 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:

$$f(x) = \sum_{j=1}^{n} \alpha_j y_j G(x_j, x) + b, \tag{5.4}$$

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.

5.1.4 The Dataset Parameters Determination



(a) Accuracy for different flow time window size.

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

Figure 5.1: Accuracy of decision tree, SVM, and neural network algorithms for the two flow combining approaches, all packets in a fixed time window size and first few packets of a flow.

We have to determine two important parameter for our threat detection architecture: the flow window-time size and the number of first packets that is used to characterize a flow. Short window-time sizes provide a faster threat detection but can compromise the accuracy. Similarly, the less is the number of first packets required to classify a flow, the faster the detection is. Figure 5.1 shows that both a 0.5 second window size and only two packets result in a low accuracy, around 70% that correspond to the most frequent class of the dataset. This accuracy result, however, improves as the architecture gathers more information about the flows. We use GTA/UFRJ dataset and the classification algorithms implemented to choose the parameters. Figure 5.1a shows the accuracy result for the approach with all packets gathered in different time windows. 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 size. We then choose the one second window size, because flow composition gathers enough information to correctly classify the samples with shortest possible time. Figure 5.1b 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 in its very start and, as the result shows, five packets are enough to obtain a high accuracy, near 99%. Therefore, we choose five packets as the most suited number of analyzed packets. Besides presenting a higher accuracy than the approach that analyzes all packets in a one second window, the approach with the five first packets shows other advantages, such as: shorter time to extract the features of a flow; greater robustness; and more potential for scalability, since this way there is no need to process a large number packets for each flow, which can be critic in flood threats.

5.1.5 Feature Selection and Principal Component Analysis

We perform dimensionality reduction, to improve the efficiency of the proposed architecture in real-time threat detection. The aim is to improve the throughput, eliminating irrelevant features of the threat detection procedure. Another important aspect is a possible correlation between two or more features, which can be combined into only one feature, reducing the processing time [16].

We achieve dimensionality reduction through the Principal Component Analysis (PCA), which transform a group of possibly correlated variables into a group of linear uncorrelated variables that lie in orthogonal planes. This transformation takes into account the eigenvalues in a way that the component associated to the highest 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 highest eigenvalues, because they have more relevant information and we withdraw the components associated to the lowest 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.

For both ways of combining packets into flows, the sum of the higher eight eigenvalues represents more than 95% of the total sum, as shown in Figure 5.2. In other words, the first eight features from the components calculated by the PCA linear transformation represent 95% of the total variance. Therefore, these eight components are selected and the others, that represents less than 5% of the total

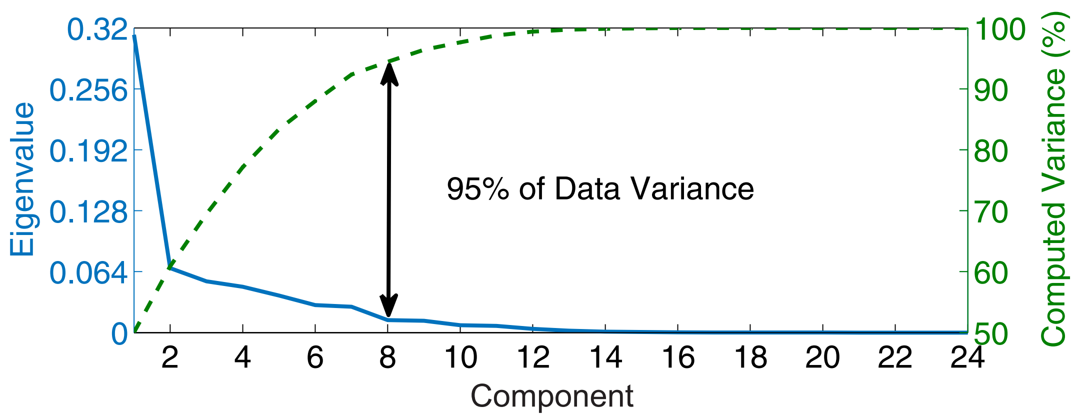


Figure 5.2: Eigenvalue for each flow feature. The eigenvalue associated to each of the transformed features is proportional to the data variance. The eight highest principal components represent 95% of the total data variance.

data variance, are discarded, improving the processing time, which is critical in real-time applications.

5.1.6 Historical based Threat Classification Results

Tables 5.1 and 5.2 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 the first five-packet approach has a higher accuracy than the one-second time window for SVM and Neural Network classification methods. The accuracy improvement is due to the behavior of applications and threats that is better defined in beginning, usually when the negotiation phase of applications happens. Both SVM and Neural Network perform well in all scenarios. SVM is a very robust classifier, since it maximizes the margin to the decision threshold, obtaining good classification results. Neural Network can adapt to complex underlying functions, because of the combination of the each neuron calculation that produces high order classification functions. As shown in Tables 5.1 and 5.2, the accuracy for both these algorithms is higher than 95% for all scenarios, ensuring the efficiency of the proposed architecture to detect known threats. The five first packets of each flow approach has better results than the one second time window, for these two algorithms, because it is harder to hide the malicious behavior with only a few packets being analyzed. We check periodically the first packets of each flow, since we consider a flow as a sequence of packets from the same source IP to the same destination IP in a specific time window. In order to avoid being tricked by attackers, our architecture randomizes this period and, therefore, the attackers cannot simulate a legitimate use and then engage an attack.

We further show the confusion matrix of SVM and Neural Network algorithms

Table 5.1: Accuracy comparison for the 3 classification methods in the GTA Dataset.

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

Table 5.2: Accuracy comparison for the 3 classification methods in the NetOp Dataset.

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

with higher accuracy for each approach. The confusion matrix clearly specifies the false positive rate, attack detection rate, and other metrics of each class in the test dataset. This matrix is in the format real class versus predicted class. The lines represent the elements that belong 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. From this metric, other metrics like false positive rate and true positive rate can be derived.

Table 5.3: GTA/UFRJ dataset: First Five Packets of Flow Neural Network Confusion Matrix.

Real \ **Predicted**	**Normal**	**DoS**	**Probe**
Normal	27899	17	32
DoS	0	5095	0
Probe	396	1	10030

Table 5.4: GTA/UFRJ dataset: One Second Time Window SVM Confusion Matrix.

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

Tables 5.3 and 5.4 show the confusion matrix for both approaches to combine flows for the GTA/UFRJ dataset. Both approaches present a very low false positive rates, lower than 2%. Tables 5.5 and 5.6 show the confusion matrix for both approaches in the NetOp dataset. For this dataset, we classify the flows only as

malware threat and legitimate usage. For both datasets, the threat detection rate is above 87%.

Table 5.5: NetOp dataset: First Five Packets of Flow SVM Confusion Matrix.

Real \ Predicted	**Normal**	**Threat**
Normal	1257069	51151
Threat	5285	133570

Table 5.6: NetOp dataset: One Second Time Window SVM Confusion Matrix.

Real \ Predicted	**Normal**	**Threat**
Normal	1424926	51681
Threat	15363	104317

5.2 Adaptive Threat Detection

We propose the use of a live feed of threat information from honeypots and the use of online trained machine learning algorithms to achieve an adaptive threat detection. Attackers keep changing their behavior and also keep looking for new vulnerabilities to mislead security systems. Zero-day threats are a challenge to current security systems since there is no information available to create signatures and update intrusion detection system databases. Our architecture address this challenge by training the models in real time as new threats arrive to the honeypot. Moreover, we also propose the use of online trained anomaly detection algorithms to detect zero-day attacks in case they are performed against the users instead of the honeypots. The proposed anomaly detection algorithms profile legitimate user behavior and detect threats by analyzing the deviation from the standard legitimate behavior. When a sample differs more than an accepted threshold from legitimate behavior, it may represent an unknown threat and triggers an alert.

5.2.1 Stochastic Gradient Descent with Momentum

The Stochastic Gradient Descent (SGD) algorithm is a stochastic approximation of Gradient Descent, in which the gradient is approximated by a single sample. In our application, we consider two classes, normal and threat. Therefore, we use the Sigmoid Function 5.5 to perform logistic regression. In the Sigmoid function, low product values of the parameters $\theta^\intercal$ times the sample feature vector x return zero, whereas high values return one.

$$h_\theta(x) = \frac{1}{1+e^{-\theta^\intercal x}} \tag{5.5}$$

When a new sample $x_{(i)}$ arrives, the platform evaluates the Sigmoid function and returns one for $h_\theta(x_{(i)})$ greater than 0.5 and zero otherwise. This decision presents an associated cost, based on the real class of the sample $y_{(i)}$. The cost function is defined in Equation 5.6. This function is convex and the goal of SGD algorithm is to find its minimum, expressed by

$$J_{(i)}(\theta) = y_{(i)}log(h_\theta(x_{(i)})) + (1-y_{(i)})log(1-h_\theta(x_{(i)})). \tag{5.6}$$

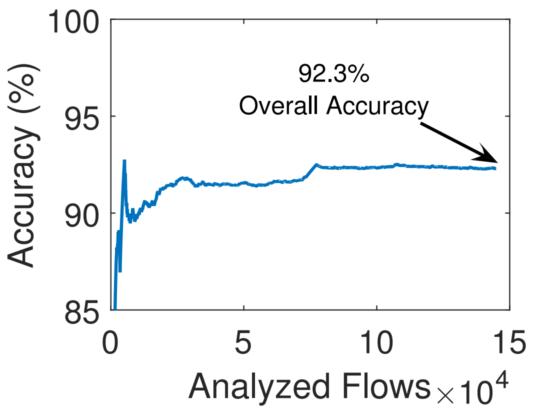
When a new sample arrives, the algorithm takes a step toward the cost function minimum based on the gradient of the cost function.

```
input  : Incoming flow features $x$, Class $y$
output: Predicted Class $predict$, training parameters $\theta$
Initialize $\theta$, $\Delta\theta$, $\alpha$, $\beta$;
for $i \leftarrow 1$ to $m$ do
    $h_\theta(x_{(i)}) = \frac{1}{1+e^{-\theta^\intercal x_{(i)}}}$;
    $predict = round(h_\theta(x_{(i)}))$;
    if $predict == 1$ and $y_{(i)} == 0$ then
        Send Alert;
    else
        $\theta = \theta - \alpha\nabla J_{(i)}(\theta) + \beta\Delta\theta$;
        $\Delta\theta = \alpha\nabla J_{(i)}(\theta)$;
    end
end
```

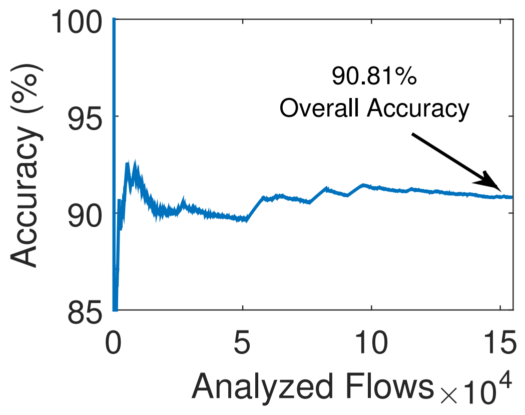
Algorithm 1: Stochastic Gradient Descend with Momentum.

Algorithm 1 shows the implemented SGD algorithm. At each incoming sample vector $x_{(i)}$, the platform determines the class $y_{(i)}$ based on the type of source. If it comes from a honeypot, the label is 1, while if it comes from a traffic analyzer the label is 0. If the sample comes from a traffic analyzer and SGD predicts it as a threat, it sends an alert. Otherwise, it updates the parameters based on the cost function gradient. The term $\Delta\theta$ is the momentum and has the value of the previous parameter update. In physics, the term momentum indicates the difficulty to change the movement of a rotating object. In SGD context, this term considers the past move when updating the parameters θ. The parameters α and β are periodically updated in an offline manner based on the historical cost for each sample. This is possible due to the feedback from the batch processing layer to the stream processing layer shown in Figure 3.1.

Figure 5.3 and 5.4 show the accuracy behavior over time for each incoming sample, and Tables 5.7, 5.8, 5.9, and 5.10 show the final confusion matrix for both

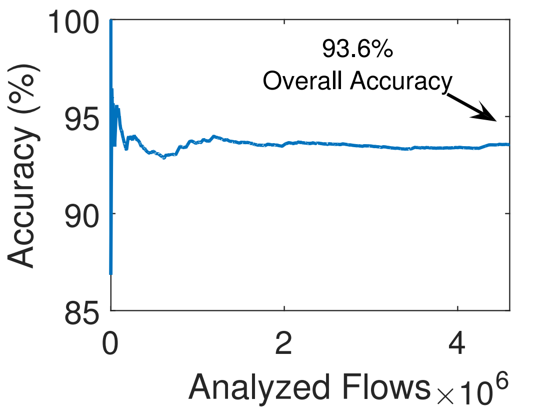


(a) First Five Packets of Flow.

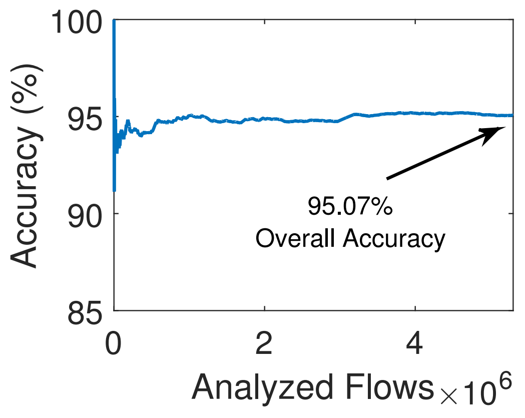


(b) One Second Time Window.

Figure 5.3: Stochastic Gradient Descend Accuracy for the GTA/UFRJ dataset. In both cases, the accuracy stays stable even with new attacks and legitimate usage behavior changes.



(a) First Five Packets of Flow.



(b) One Second Time Window.

Figure 5.4: Stochastic Gradient Descend Accuracy for the Network Operator Dataset. Once again, even the accuracy stays stable over time, what demonstrate great adaptability.

datasets and both approaches for extracting flow features. At the very beginning of the analysis, during the setup time, overfitting occurs because the algorithm has few samples and adapts specifically to them, resulting in very high accuracy. However, as samples arrive, the SGD stabilizes and acquires great capability to generalize and adapt to new traffic samples, since the accuracy does not present great variations, ending with accuracies of 92.3% and 90.81% for the GTA/UFRJ dataset with the one second flow window size and the first five packets of each flow respectively. For the NetOp dataset the accuracy results are 93.6% and 95.07% for the same approaches of extracting features. Despite threat detection rates of 79.5% and 76.19% for the GTA/UFRJ dataset and 54.3% and 65.66% for the NetOp dataset, respectively for the first five packets of each flow and one second time window, this algorithm has very low false positive rates of 1,2% and 2.74% for the GTA/UFRJ dataset 0.51% and 0.53% for the NetOp dataset. The low false positive rate is a well-desired characteristic, since it ensures confidence in the platform security alerts.

Table 5.7: SGD Confusion Matrix for the First Five Packets of Flow GTA/UFRJ Dataset.

	Normal	**Threat**
Normal	94927	1147
Threat	10010	38817

Table 5.8: SGD Confusion Matrix for the One Second Time Window GTA/UFRJ Dataset.

	Normal	**Threat**
Normal	104028	2927
Threat	11245	35987

Table 5.9: SGD Confusion Matrix for the First Five Packets of Flow NetOp Dataset.

	Normal	**Threat**
Normal	4172989	21431
Threat	287441	341722

Table 5.10: SGD Confusion Matrix for the One Second time Window NetOp Dataset.

	Normal	**Threat**
Normal	4603972	24627
Threat	237739	454617

5.2.2 Online Support Vector Machine

The Support Vector Machine (SVM) is a binary classifier, based on the concept of a decision plane that defines the decision boundaries. A hyperplane constructed in a multidimensional space splits the data into classes. The online SVM algorithm uses a soft margin approximation with the convex hinge-loss function, given by

$$\max\{0, 1 - y\theta^\intercal x\}. \tag{5.7}$$

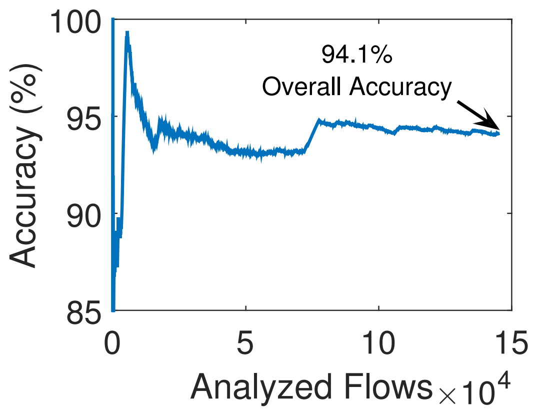
The objective of this algorithm is to minimize the loss function. Just like the previous algorithm, the platform determines the class $y_{(i)}$ based on the source. If it comes from a honeypot the label is 1, while if it comes from a traffic analyzer the label is -1. Again, when a traffic from a network sensor is classified as threat, the platform sends an alert and does not update the model. Algorithm 2 shows the implementation of online SVM. The parameters α, λ are periodically updated based on the evaluation of the hinge-loss function over the samples.

Figures 5.5 and 5.6 show the accuracy changes over time at each incoming sample, and Tables 5.11, 5.12, 5.13, and 5.14 show the final confusion matrix for both datasets and both feature extraction approaches. The overall accuracy is better than the SGD algorithm, because the SVM is a robust classifier that maximizes the margin to the hyperplane decision boundaries. Figures 5.5 and 5.6 also show that the algorithm adapts really well to usage changes and new attacks, maintaining the high

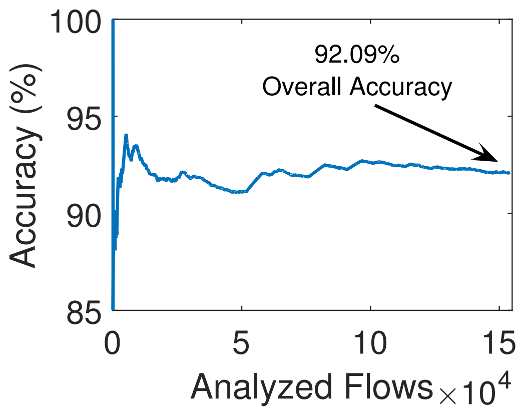
```
input  : Incoming flow features $x$, Class $y$
output: Predicted Class $predict$, training parameters $\theta$

Initialize $\theta$, $\alpha$, $\lambda$;
for $i \leftarrow 1$ to $m$ do
    $predict = sign(\theta^{\intercal} x_{(i)})$;
    if $predict == 1$ and $y_{(i)} == -1$ then
        Send Alert;
    else
        if $y_{(i)} \theta^{\intercal} x_{(i)} > 1$ then
            $\nabla_{(i)} = \theta$;
        else
            $\nabla_{(i)} = -\lambda y_{(i)} x_{(i)}$;
        end
        $\theta = \theta - \alpha \nabla_{(i)}$
    end
end
```

Algorithm 2: Online Support Vector Machine.

accuracy. The overall accuracies are 94.1% and 92.09% for the GTA dataset and 95.6% and 97.22% for the NetOp, considering the first five packets and one second time window approach. As the confusion matrices show, the attack detection rate is higher than SGD, with values of 90,1% and 87.51% for the GTA/UFRJ dataset and 81.5% and 82.06% for the NetOp dataset, respectively to the first packets of each flow approach and the time window approaches. In addition, SVM also gets a good false positive rate, with values of 3.9% and 5.89% for the GTA/UFRJ dataset and 2.26% and 0.50% for the NetOp dataset.



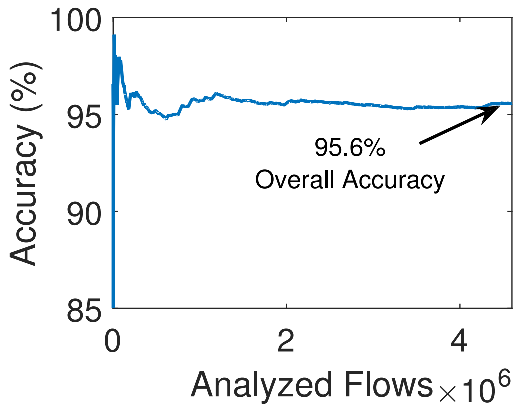
(a) First Five Packets of Flow.



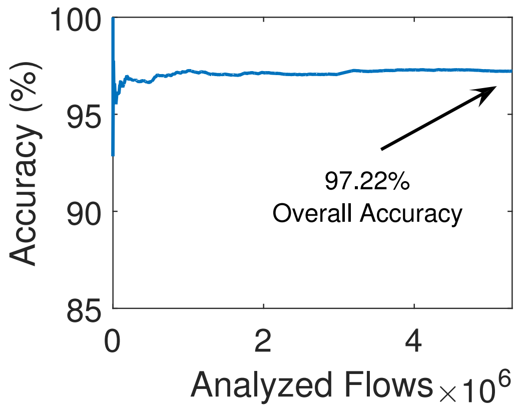
(b) One Second Time Window.

Figure 5.5: Online Support Vector Machine Accuracy for the GTA/UFRJ dataset. Again, even with behavior changes, the accuracy stays stable.

The Online SVM presents an overall accuracy and attack detection rate higher than the Stochastic Gradient Descent with Momentum. Due to the maximization



(a) First Five Packets of Flow.



(b) One Second Time Window.

Figure 5.6: Online Support Vector Machine Accuracy for the NetOp dataset.

Table 5.11: SVM Confusion Matrix for the First Five Packets of Flow GTA/UFRJ Dataset.

	Normal	**Threat**
Normal	92359	3715
Threat	4834	43993

Table 5.12: SVM Confusion Matrix for the One Second time Window GTA/UFRJ Dataset.

	Normal	**Threat**
Normal	100658	6297
Threat	5900	41332

Table 5.13: SVM Confusion Matrix for the First Five Packets of Flow NetOp Dataset.

	Normal	**Threat**
Normal	4099766	94654
Threat	117051	512112

Table 5.14: SVM Confusion Matrix for the One Second time Window NetOp Dataset.

	Normal	**Threat**
Normal	4605272	23327
Threat	124430	567926

of the separation hyperplane margin, SVM achieves a greater ability to classify the threats that presents fewer samples in both datasets. However, SGD has a lower false positive rate. Thus, when considering the security constraints, SVM would be recommended when the network requires a higher threat detection and SGD for networks that have lower security constraints and privilege user experience, therefore not allowing many false positive alerts.

5.2.3 Threat Classification Summary

Table 5.15 shows accuracy for all proposed threat classification methods. Moreover, this Table details precision and recall for both legitimate and malicious traffic. Precision and recall are common metrics for evaluating the results for a specific class. Since the main 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 show the number of true and false alerts that would trigger counter measures.

Table 5.15: Threat classification summary for all algorithms and datasets. We show accuracy, 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**	**Accuracy**	**Normal**		**Threat**	
			Precision	**Recall**	**Precision**	**Recall**
Decision Tree	GTA 1s	80.6%	80.7%	94.6%	80.1%	48.8%
	GTA 5p	99.9%	99.9%	99.9%	99.9%	99.9%
	NetOp 1s	92.8%	97.3%	94.8%	51.5%	67.9%
	NetOP 5p	86.3%	99.1%	85.6%	40.6%	92.9%
Neural Network	GTA 1s	96.0%	94.8%	98.8%	97.9%	91.5%
	GTA 5p	99.0%	98.6%	99.8%	99.7%	97.4%
	NetOp 1s	95.1%	97.5%	97.2%	66.7%	69.2%
	NetOp 5p	95.3%	98.2%	96.6%	72.2%	83.1%
SVM	GTA 1s	96.3%	95.4%	98.8%	98.0%	92.4%
	GTA 5p	98.6%	98.0%	99.7%	99.5%	96.4%
	NetOp 1s	95.8%	96.5%	98.9%	66.9%	87.2%
	NetOp 5p	96.1%	99.6%	96.1%	72.3%	96.2%
SGD	GTA 1s	90.8%	90.2%	97.3%	92.5%	76.2%
	GTA 5p	92.3%	90.5%	98.8%	97.1%	79.5%
	NetOp 1s	95.1%	95.1%	99.5%	94.9%	65.7%
	NetOp 5p	93.6%	93.6%	99.5%	94.1%	54.3%
Online SVM	GTA 1s	92.1%	94.5%	94.1%	86.8%	87.5%
	GTA 5p	94.1%	95.0%	96.1%	92.2%	90.1%
	NetOp 1s	97.2%	97.4%	99.5%	96.1%	82.0%
	NetOp 5p	95.6%	97.2%	97.7%	84.4%	81.4%

For any given class, the precision is the fraction of correctly classified samples over all samples predicted to belong to such class, while the recall is the fraction of correctly classified samples over all samples that really belong to that class. As Table 5.15 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 absolute number of normal samples classified as threat, they have a negative impact on the precision. However, 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 batch trained algorithms have better recall when detecting threats. Batch training can consider all samples at a time when adjusting the parameters, therefore achieving a better result. A downside of this approach, however, is the detection of zero day attacks, because they have to be retrained to learn and correctly classify the new threats, while the online trained algorithms can adapt in real time.

For both batch and online trained algorithms, SVM presents the most constant behavior, maintaining high accuracy, precision, and recall results. SVM is well known for its robustness. Although there are difference between the online and batch implementations, both SVM approaches try to maximize the margin between the hyperplane and the samples that belong to each class. As Table 5.15 shows, in all scenarios the batch SVM achieve above 87% in the recall for the threat class, representing good threat detection capabilities, while maintaining low false alert generation. The same behavior is observed in the online SVM with threat recall above 81% and low false alert generation. Moreover, the online SVM is also able to detect new threats as the data arrives from the honeypots.

5.2.4 Anomaly Detection by Normal Distribution

We argue that protection against unknown attacks is essential to have a higher level of security in computer networks. Nevertheless, zero-day attacks are hard to detect, since there are not yet any previous information about the attack. Anomaly detection is capable to discover new attacks. 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 identified 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.

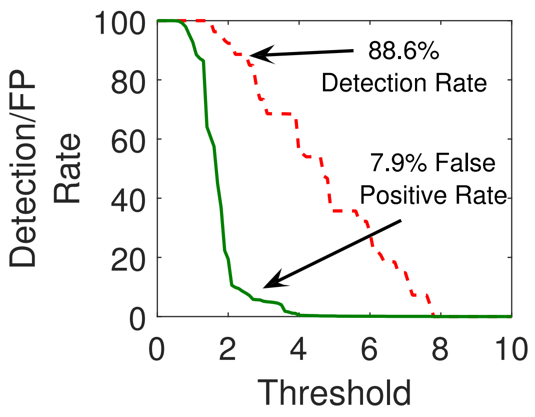
The real-time implementation requires the anomaly detection as the streaming data is arriving. The anomaly is detected if at least one of the following conditions is true for at least one feature j, taking into account the means μ_j and the variances σ_j^2 calculated in training:

$$X_j > \mu_j + threshold * \sigma_j^2 \qquad (5.8) \qquad X_j < \mu_j - threshold * \sigma_j^2 \qquad (5.9)$$

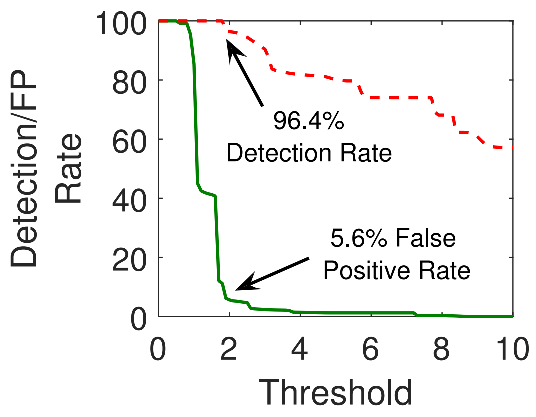
The proposed architecture allows real-time anomaly training. Consequently, the algorithm becomes adaptive, which is fundamental for anomaly detection, since the network behavior may change in time. Therefore, when a new sample arrives and it is not detected as an anomaly by conditions (5.8) and (5.9), 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:

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} X_j \qquad (5.10) \qquad \sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(X_j - \mu_j)^2 \qquad (5.11)$$

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. Consequently, the normal distribution parameters are always updated by samples considered legitimate, ensuring adaptability.



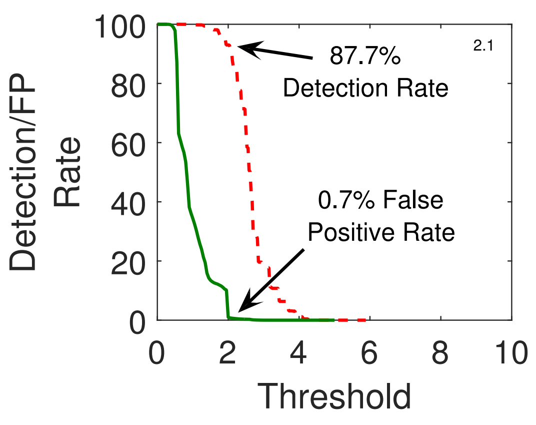
(a) First Five Packets of Flow.



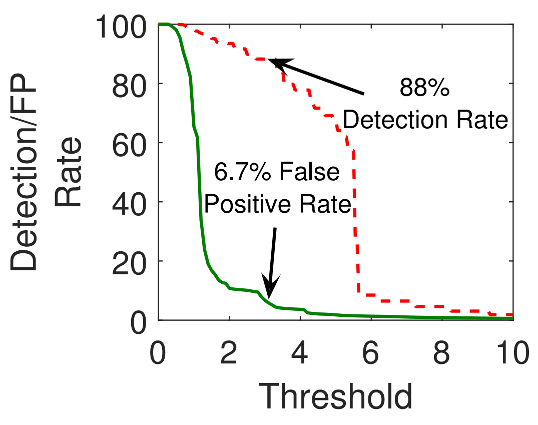
(b) One Second Time Window.

Figure 5.7: 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.

Figure 5.7 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, almost all threats are detected, however at the cost of a high false positive rate. On the other hand, a high threshold value results in less false



(a) First Five Packets of Flow.



(b) One Second Time Window.

Figure 5.8: False positive and attack detection rates for the NetOp dataset according to the threshold.

positives, but also in 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 threshold parameter must be chosen depending on the application and considering a trade-off between the false positive and attack detection rates.

Figure 5.8 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.

5.2.5 Anomaly Detection by Entropy Time Series

We implement another anomaly detection method, by analyzing the entropy value of a sliding window of flows. The sample entropy indicates the degree of concentration or dispersion of a feature. It is calculated as follows:

$$H(X) = -\sum_{i=1}^{N} \left(\frac{n_i}{S}\right) \log_2\left(\frac{n_i}{S}\right) \tag{5.12}$$

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 all values are concentrated in one range, $H(X)$ 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 entropy values, equally distributed. Another parameter determined in the training phase is the range with the most samples. We observed that the normal traffic entropy values tend to be concentrated together and that usually the most frequent value is in the middle of this concentration. Thus, we propose the detection 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 networks behaviors.

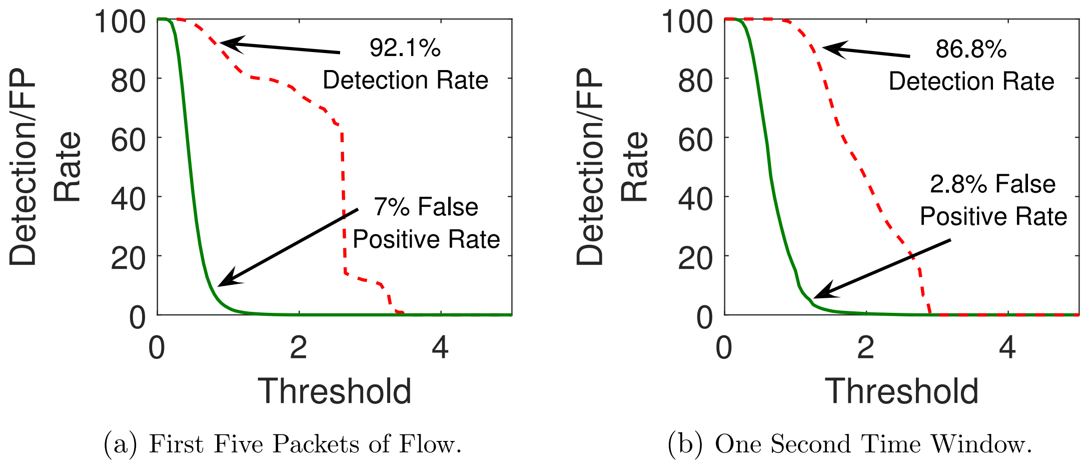


Figure 5.9: False positive and attack detection rates for the GTA/UFRJ dataset according to the entropy threshold. The threshold represents the distance to the range that has the most entropy samples.

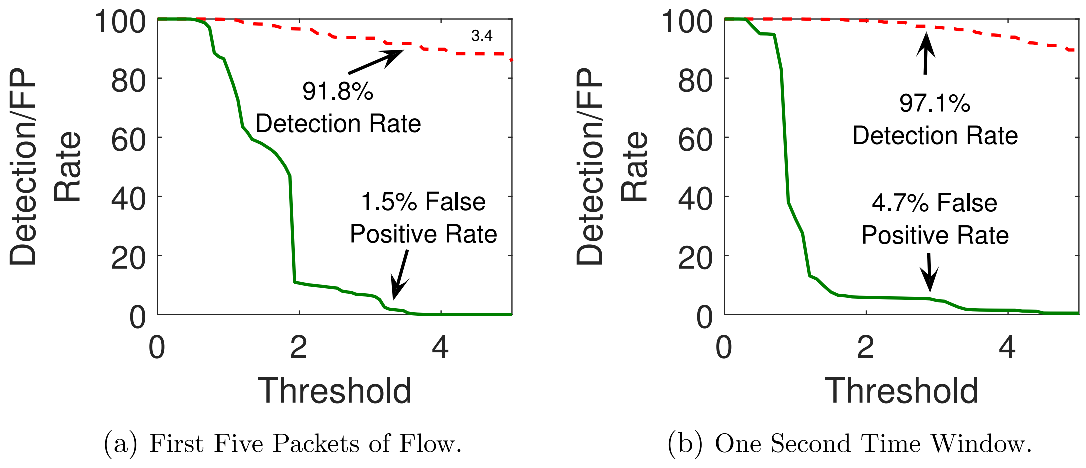


Figure 5.10: False positive and attack detection rates for the NetOp dataset according to the entropy threshold. The threshold represents the distance to the range that has the most entropy samples.

Figure 5.9 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 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 5.10 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 very 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 a very good trade-off in all scenarios.

Chapter 6

The Intrusion Prevention Architecture

Intrusion prevention systems are deployed in two modes, on-path and off-path. In the on-path mode, the analysis is performed in the network traffic route from the source to its destination. In the off-path mode, the packets are mirrored and sent to an analysis that will determine whether the packet is part of a threat. The great advantage of the on-path mode is the ability to perform traffic block, since the traffic is analyzed and then forwarded to its destination. However, this approach introduces latency problems, considering that the analysis is introduced before the forwarding procedure. On the other hand, the off-path mode allows gathering information from other sources, thus, the system can correlate important data to improve detection accuracy. However, the off-path mode needs an additional infrastructure to carry both the traffic mirroring and threat blocking. In our proposal, we choose the off-path detection mode and we propose an entire scheme to efficiently mirror and block threat traffic using the programmability of Software Defined Networking (SDN).

We perform the attack detection based on information from the first five packets of each flow, since the performance results of the flow defining approach that analyzes the first few packets are better than the other approach. Whenever an undefined flow arrives in a SDN switch, a message is forwarded 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 is under analysis and when that number reaches five packets, this machine sends the flow information to the processing module and also a message warning the controller that the analysis of the packets from that particular flow has been completed. A flow is defined as all packets from the same IP source and IP destination during a time window 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

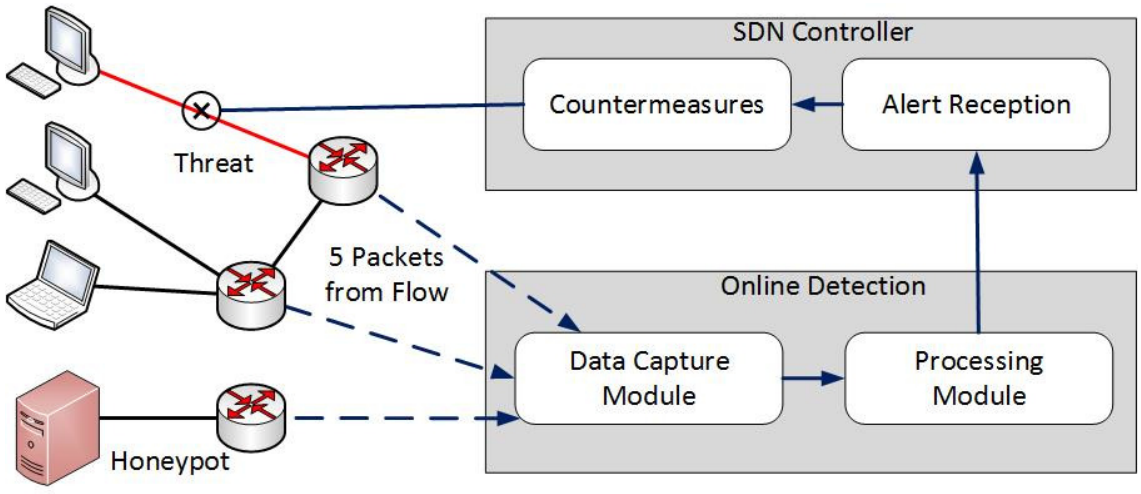


Figure 6.1: Example of network monitored by the proposed architecture. The network controller installs the mirroring rules to the capture module that sends traffic information to the processing module. Through alerts from the threat detection, the network controller blocks threats.

can consume 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 analysis machine, making the detection and prevention process robust, because the analysis machine is protected against flood attacks and has a greater ability to analyze flows when compared with the all packets analyzing approach. It is important to remark that when the controller removes the mirroring action, the action will again be installed after the flow timeout set in the controller, which makes the flows to be periodically checked, increasing network security. This timeout is randomly chosen within a range of values, to avoid that an attacker tricks the architecture, by setting legitimate flows during the beginning of the analysis.

Figure 6.1 shows an example network analyzed and protected by the proposed Threat Detection and Prevention Architecture [12]. All SDN switches have an interface to which traffic is duplicated and sent to a analysis machine. The controller is responsible for installing the mirroring rule and for removing it when the machine has received the five required packets. The data capture module, composed by the analysis machines, then forwards the flows features to the detection application, which utilizes batch and online trained machine learning algorithms in conjunction with stream processing in real time to do its analysis. If a threat is detected, the detection architecture sends an alert to the controller, which then blocks the attacker source IP in all switches.

6.1 Defense Strategy against Spoofed Source IP Threats

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, since, besides the attack damage, they overload the controller to set new flows every time the source IP changes, generating also a large number of entries in the switches flow table.

To solve this problem, 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 so, an alert arrived soon after, it may indicate that the blocking rule was not effective. Therefore, the controller keeps track of time between the alerts reception and notices when two alerts arrive in a very 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 the flows of the network for a certain period of time. This is possible due to the network global view of the controller, which knows the entire network topology. Then, if a third alert 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 to the network and block this port. Thus, besides the IP blocking rule, it also instantiates a blocking rule of the traffic from the attacker port. Here we block the traffic from the port considering that the intruder is in the SDN network. However, other actions are applicable, such as transfer the traffic to a honeypot or traffic filter, and limit the bandwidth.

Therefore, if an attacker spoofs its source IP, after three alerts, the attack traffic is blocked. The additional processing by the controller to monitor the path of all flows only starts when two alerts arrive in a short period. After a period of time without receiving any new alert, the process is undone, avoiding the waste of resources. Another important aspect is to maintain IP blocking rule for the attacker, because it cannot be an attack with spoofed IP, but a distributed attack. Against a distributed attack, it is essential to block all sources and when preventing traffic from both inbound interface on the network and the source IP, the blocking rule will be effective.

Algorithm 3 summarizes the defense strategy against spoofed IP addresses. The controller has two states: normal, when there is no suspicion that an attack with spoofed IP is occurring; and monitoring, when there is such suspicion 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 consecu-

```
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 3: Defense strategy against attacks with spoofed source IP

tive alerts. When this difference is above a threshold, the controller either keeps the monitoring status or goes back to the normal status. A similar behavior occurs when this difference is below the threshold, however, the controller changes to or keeps the monitoring status. An important 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 the 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. However, other counter measures are applicable, such as reducing the bandwidth or forwarding to a honeypot or network filter.

6.2 Traffic Monitoring and Threat Block

In this section, we show the operation of the traffic analysis and the threat block scheme. For this purpose, we developed a prototype, in a real network environment, a virtual test network on a platform for experimenting future network proposals, the *Future Internet Testbed with Security* - (FITS) platform [54]. We implemented this prototype to evaluate the proposed architecture, including the monitoring scheme of the five packets of each flow, and traffic blocking using SDN, even in scenarios with spoofed IP addresses.

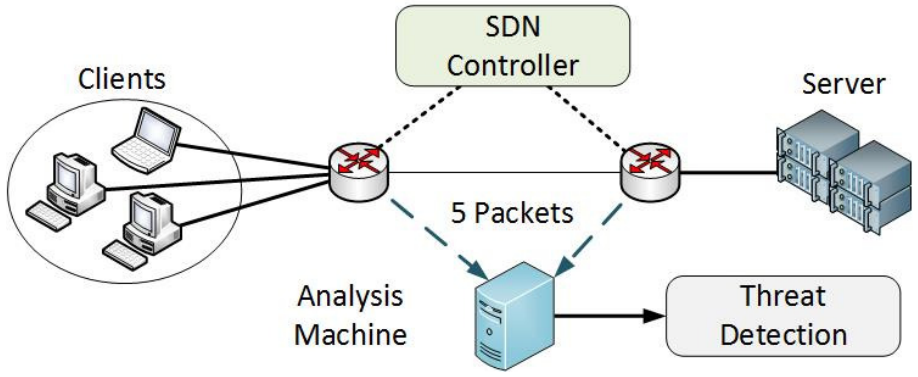
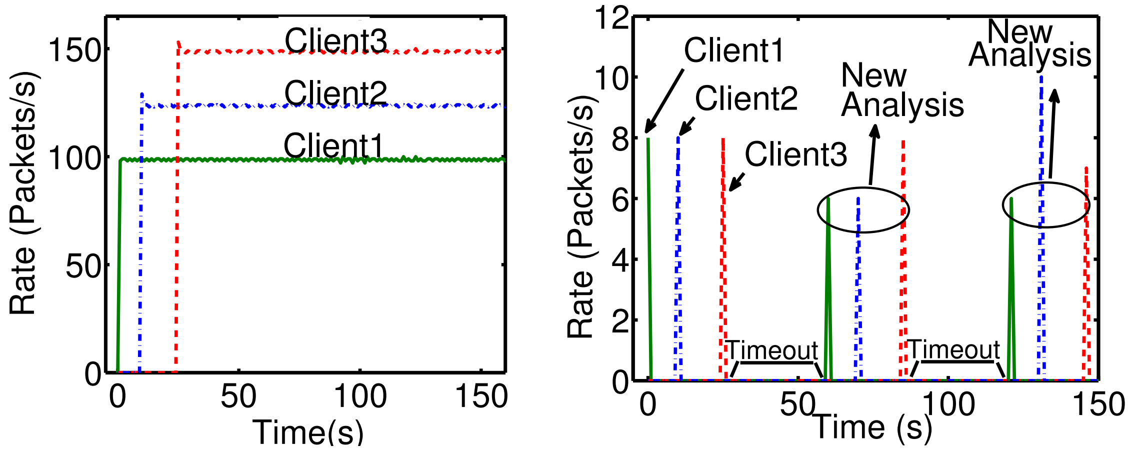


Figure 6.2: Experimental network topology used for the five packets scheme and the malicious traffic block.

In this experimentation platform, we use Xen hypervisor for virtualization and OpenFlow for traffic forwarding. Figure 6.2 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 extract flow characteristics using `Bro`. The connection between switches and the analysis machine requires a Generic Routing Encapsulation - (GRE) tunnel, which encapsulates the packets and sets the address of the analysis machine as destination. The analysis machine decapsulates the packets and then analyzes it, extracting the flow characteristics to send them to the detection architecture. We perform the experiments on an Intel Xeon X5690 server with 24 processing cores, each of them with a frequency of 3.47 GHz clock and with 48 GB of RAM.

The results of the first experiment, shown in Figure 6.3, aim to display the operation of the traffic deviation 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 6.3a shows the traffic received by the server machine. We can observe in the figure that the proposed scheme does not affect communication. Furthermore, Figure 6.3b shows the packet rate received by the analysis machine, which sends a message to the network controller after the capture of the five packets needed to characterize a flow. Even though the sending rate is much higher, the analysis machine receives few packets. The reason the analysis machine receives a little more than five packets is time required to inform the controller to undo the deviation rule. In addition, this figure shows that flows are analyzed periodically, accordingly to the flow timeout set in the controller. In this experiment, the flow timeout was set within 60 seconds, and, therefore, the analysis machine receives the packets of each client at every minute.



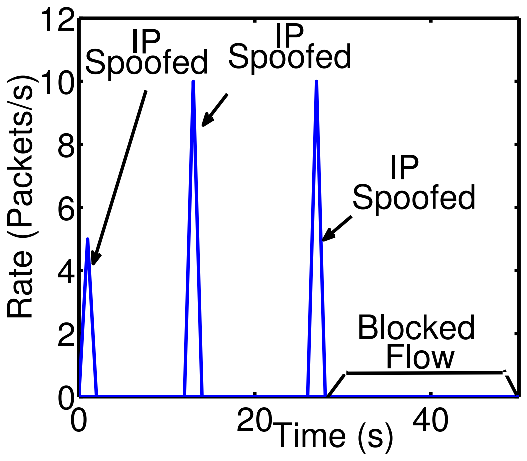
(a) Packet rate received by the server machine.

(b) Packet rate received by the analysis machine.

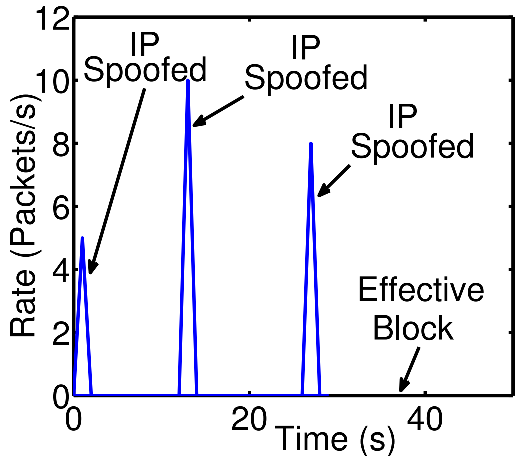
Figure 6.3: Network operation when three machines communicate with the server. The server receives traffic without being affected by the proposed scheme, while the analysis machine only receives the first packets of each flow.

Two important aspects of this proposal are the time required to characterize the flow and the ability to increase the number of flows to be analyzed. Since 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 a shorter detection time and, thus, a faster threat block. Furthermore, the analysis machine is immune to flooding attacks because not all packets of a flow are deviated, and for this reason, the analysis machine can receive a much larger number of flows, ensuring robustness and potential for scalability.



(a) Packets rate of attacker received by the server machine.



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

Figure 6.4: 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 6.4 shows the traffic received by the server and the analysis machine. After the attack started and the analysis machine sends the information to the Detection Module, a rule blocking the source IP is immediately installed on all switches, which makes the traffic received by the server, shown in Figure 6.4a, remain zero after the detection time. However, when the attacker changes the IP for the first time, the flow is once again analyzed and blocked as shown in Figure 6.4b. Nevertheless, this time, when the controller receives the alert, it starts to mark the path of all flows, so it can map in which port each flow enters the network. Once again, around 28 seconds, the attacker changes the source IP and, after the analysis, a new alert is generated. This time though, when the controller receives this alert, it blocks both the source IP and the source port in which this flow enters the network. Here we choose to block the traffic from the port in which the attacker enters the network. However, other policies are applicable, such as redirect the traffic to a filter out the malicious flow. From this point on, when the attacker changes the source IP, the traffic is not even analyzed again, since it is blocked as near to its origin as possible.

6.3 Providing Elasticity to Sensor Elements

We propose an elastic architecture for threat detection in virtual networks, with dynamic allocation of virtual sensor elements according to the demand [10]. In addition to the sensor element creation and destruction, the architecture also provides a mechanism for redefining flows to balance targeted traffic between machines that perform traffic analysis. The proposed intrusion detection architecture consists of three main interconnected modules to control the network: the traffic characterizer module, the flow management module, and the resource management module. Figure 6.5 shows an example of a network with the proposed elastic architecture. Network traffic is mirrored at the switches for sensor element virtual machines. There is a communication between the controller and the physical machines to send statistics regarding the resource consumption and the number of analyzed packets, to allow the mirroring rule to be disabled.

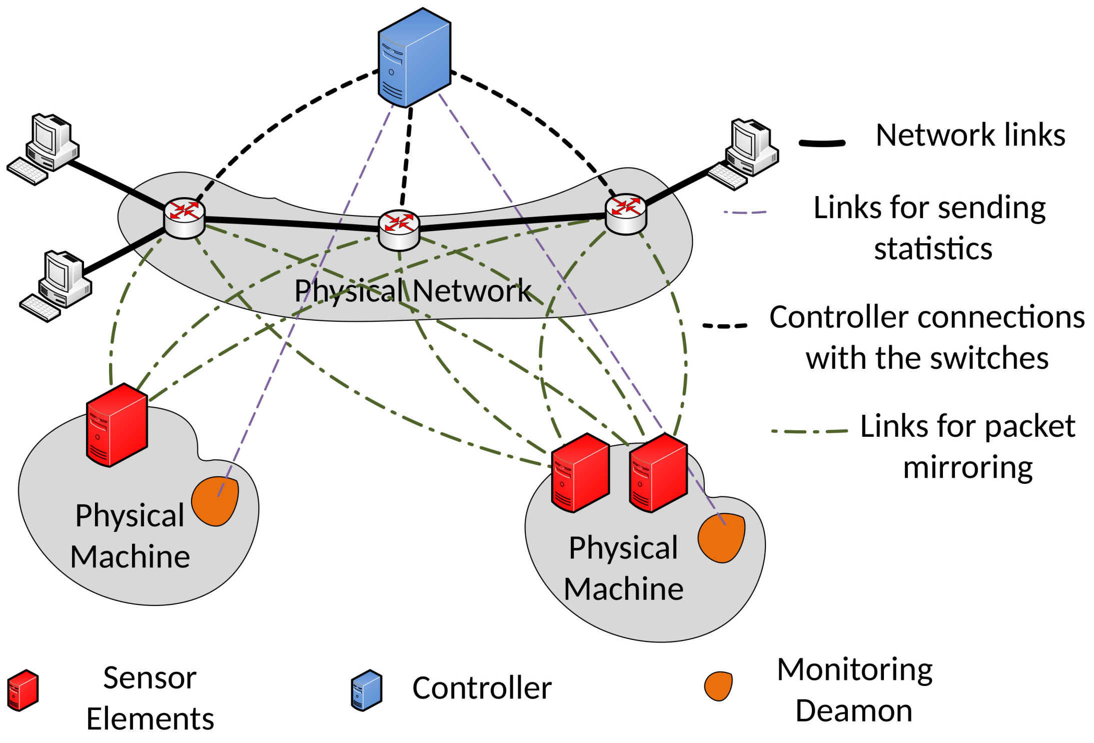


Figure 6.5: In the proposed architecture, a physical machine may contain several sensor elements. The controller connected to the switches can redefine flows taking into account the resource consumption statistics of the sensor virtual machines.

The traffic characterizer module is composed by virtual machines that extract flow features and send them to the `Kafka` publish/subscribe platform. These features are then consumed by the real-time stream processing threat detection application that sends alerts to the controller. In the proposed architecture, `Bro` is responsible for analyzing the traffic. Each virtual machine contains a Bro sensor and uses one processing core. The traffic characterizer module adapts itself to the demand. The number of `Bro` virtual machines vary accordingly to the number of flows, being dynamically allocated. Therefore, the architecture has elastic behavior.

The flow management module is responsible for the flow distribution among the sensor elements and for executing counter measures against the attacks once the detection application sends an alert. This module is composed by a controller application that communicates with both the traffic characterizer module and the resource management module. A GRE (Generic Routing Encapsulation) tunnel mirrors the traffic to the sensor elements and the feature extraction is performed after the packet decapsulation to guarantee that the packets are not modified. Using a GRE tunnel also allows the sensor elements to be in a different network than the production one.

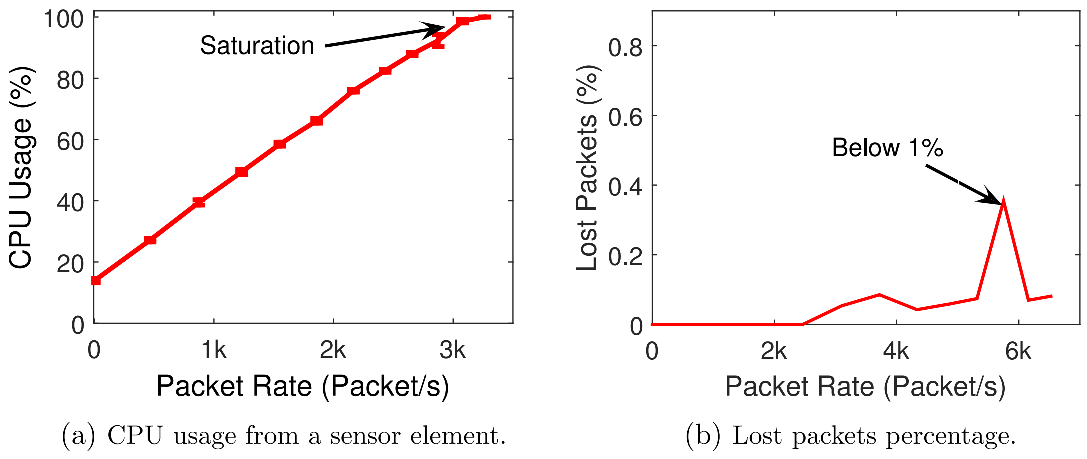
When the traffic characterizer module has more than one virtual machine, the flow distribution takes into account two aspects: the amount of resources available in each sensor element and the packet source. A flow from a new source is allocated to the machine with more processing resources available at the time. On the other hand, flows from the same source have priority to be analyzed by the same sensor, to avoid that an attack passes unnoticed.

Finally, the resource management module is present in the privileged domain of all physical machines in our proposed architecture. This module monitors the resource consumption of these machines. This module collects processing, memory, and network consumption metrics from Xen using Libvirt that is an open source API for managing virtualization platforms [55]. Each machine runs a deamon that collects physical and virtual machine statistics that are grouped in the controller. This way, the controller knows the number of sensor virtual machines available and the resource consumption from each one of them.

The architecture analyzes data from sensor elements to avoid both overload and idle resources. When an overload is detected, the resource management module analyzes the available resources among the physical machines and decides where to instantiate a new sensor element. Similarly, this module monitors the sensor elements to detect when a flow redistribution may allow shutting down one of the sensor virtual machine.

6.3.1 Resource Consumption Evaluation

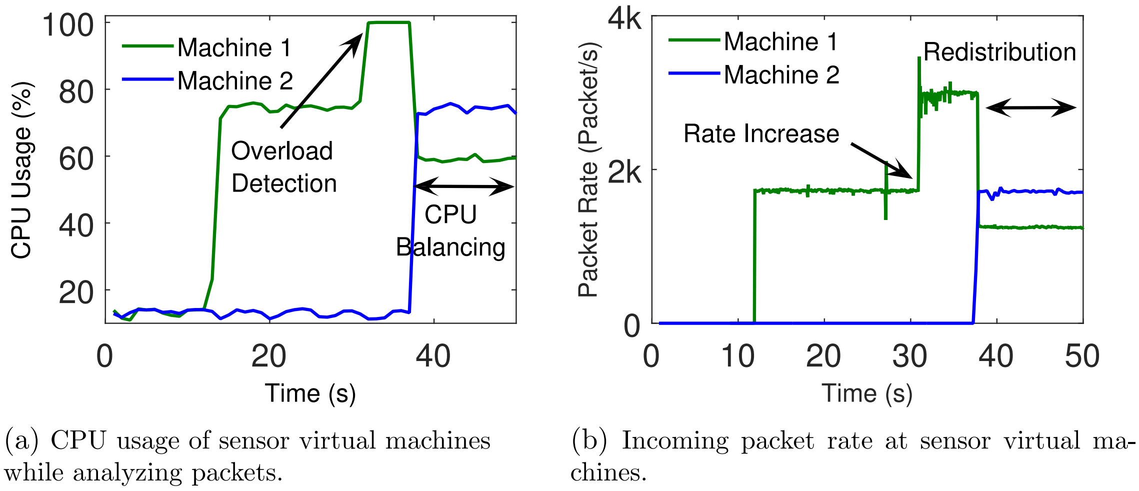
Sensor elements captures packets mirrored from the network, group them into flows, and extracts features in real time. This application consumes physical machine resources and, therefore, is subjected to resource availability. We evaluate the resource consumption of this task, in terms of processing throughput and network bandwidth that are the most critic resources in the scenario where a sensor analyzes many flows. In this experiment, we increase both the packet rate and the number of flows to evaluate the CPU usage and the percentage of lost packets. Therefore, we evaluate which resource is most relevant to sensor elements. Figure 6.6 shows the results. The sensor element uses all its CPU resources, delaying the feature extraction. However, less than one percent of incoming packets are lost, showing that processing throughput is the most relevant resource for this task.



(a) CPU usage from a sensor element. (b) Lost packets percentage.

Figure 6.6: CPU usage and lost packets percentage of a sensor element. The CPU usage achieves its limit, while the sensor loses less than one percent of the packets.

6.3.2 Sensor Element Overload

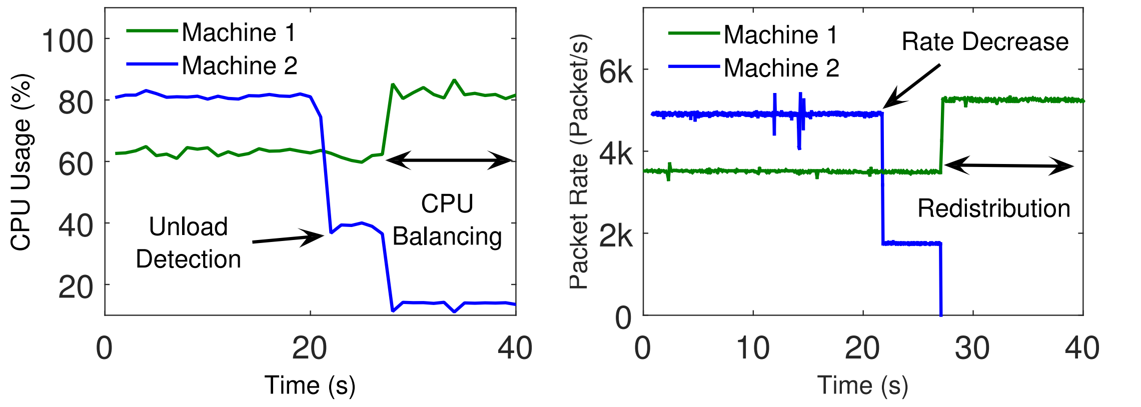


(a) CPU usage of sensor virtual machines while analyzing packets. (b) Incoming packet rate at sensor virtual machines.

Figure 6.7: Overload scenario. The CPU usage of a sensor element gets to 100%, Figure 6.7a, when the packet rate increases, Figure 6.7b. To ensure that all flows are analyzed, a new sensor machine is activated and the packets distributed.

We evaluate the behavior of the proposed architecture in a traffic characterizer module overload scenario, which occurs when a high packet rate arrives at the sensors to be grouped into flows and analyzed. In this experiment, we generate flows with constant packet rate that are analyzed by the only active sensor, due to low demand. Then we increase the packet rate, resulting in an overload of the active sensor virtual machine. When the physical machine detects this processing overload, it sends a message to the controller to inform the overload scenario. When the controller receives this message, it runs the balancing algorithm to determine in which physical machine a new sensor element should be instantiated. After the new sensor element is running, the flows are distributed, considering their source and the available resources in each machine. The results are shown in Figure 6.7. Once we increase the packet rate, an overload occurs, because the active sensor runs using the entire available processing throughput. Then, the overload is detected, and after the up time for a new sensor element, the packets are redistributed and both sensor virtual machines can processes the incoming packets.

6.3.3 Sensor Element with Idle Resources



(a) CPU usage of sensor virtual machines while analyzing packets.

(b) Incoming packet rate at sensor virtual machines.

Figure 6.8: Unload scenario. The CPU usage of a sensor element decreases and can be allocated to the other active sensor. The controller then redistributes the packets and one of the machines is deactivated.

To ensure elasticity, sensor elements must be deactivated when there are idle resources. In some scenarios, a packet distribution can leave a sensor machine without any packets to analyze, hence, allowing its deactivation and avoiding the waste of resources. We monitor constantly the resource consumption and inform resource consumption statistics to the controller that can detect when a redistribution may release one of the virtual machines. The experiment starts with two active sensor elements extracting flow characteristics, as shown in Figure 6.8. After some time, the packet rate decreases and the CPU usage of one of the sensors decreases in a way that the other sensor can analyze the remaining flows. The controller detects this decrease and redistributes the remaining analyzed packets from this machine to the other active sensor. Once one of the virtual machines does not analyze any packet, it is deactivated, avoiding idle resources.

Chapter 7

Conclusion

This work proposes a fast and efficient Threat Detection and Prevention Architecture. The threat detection is performed using machine learning algorithms combined with stream processing, while the threat prevention is based on a Software Defined Networking schema. Threat detection and prevention time is of the essence, since if the attacker passes unnoticed or no action is performed against the threat in viable time, irreparable damages will occur. This work addresses this problem by using accurate detection methods in real time, based on stream processing technology. We achieve accurate threat detection promptly after it occurs. After threat detection, a counter measure is immediately triggered by sending an alert to the controller that immediately blocks the threat source even in scenarios with spoofed IP addresses.

Another important issue to address is availability, adapting to different demand rates. In our architecture, the parallelism of the stream processor, core of the threat detection, can be increased to adapt to the demand. We can allocate more processing cores to achieve higher processing rates up to 15 Million samples per minute. Moreover, we also consider the use of multiple sensor elements to extract flow features. We monitor the resource consumption to detect overload and unload scenarios, in order to activate or deactivate sensor elements considering the demand. Therefore, we consider resource allocation in the two most critical parts of threat detection, feature extraction and machine learning detection. These applications need more resources when the network usage is increased, in order to keep threat detection running, analyzing all incoming traffic. Hence, our architecture has high availability, ensuring that traffic is analyzed and that no threat passes unnoticed.

We created a security dataset, containing real network traffic along with network threats, to evaluate the detection methods. 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. The use of two distinct datasets proves that the proposed detection methods work well even in distinct scenarios. In addition, we compare two methods of combining incoming packets into flows. In the first one, we gather all packets in a fixed length time window, whereas in the second, we periodically analyze only few packets of each flow. The advantage of the second approach is that the extracted features can be analyzed before the end of the time window, therefore speeding up the threat detection.

The proposed Threat Detection Architecture combines adaptive online trained methods with offline accurate batch trained methods. Offline batch trained algorithms tend to be more accurate, when considering known threats, because it has all available data at once and can optimize parameters considering all data items. However, this methods has an important downside, it does not detect unknown threats, because it cannot adapt in real time to behavior changes. We implemented a feedback by periodically retraining those algorithms, but since batch training requires time, it is impossible to retrain in real time. To solve this problem and detect unknown threats in real time we propose the use of a live feed of honeypot collected data to train online classification algorithms. In online training, the classifier parameters are updated considering each incoming sample, therefore adapting itself to detect zero-day attacks performed against honeypots and to legitimate usage changes. Moreover, we monitor and model legitimate usage behavior to perform anomaly detection. These methods protect the network in case a zero-day attack is performed against the network first, instead of against a honeypot. In those algorithms, we model user behavior and detect threats when a sample deviates more than an acceptable threshold from the expected behavior.

We implement seven algorithms to detect known and unknown attacks. Three of them are trained offline and are able to detect threats in real time thanks to the lambda architecture. These algorithms are Decision Tree, Neural Network and SVM. The results show a very high accuracy, above 95% for most cases and that when using the first few packets, with periodically analysis, we achieve greater accuracy. The other four algorithms represent the adaptive threat detection, that considers the online trained classification algorithms, based on honeypot data, and the anomaly detection, based on legitimate user behavior. We implemented two algorithms for each of these classes. The online classification algorithms also achieve high accuracy, always above 90%. As results show, over time, their accuracies stay stable, therefore adapting to changes in incoming traffic. We implemented the Stochastic Gradient Descent with Momentum and the Online Support Vector Machine based on honeypot data. We propose two anomaly detection methods, also online trained: one based on the Normal Distribution; and the other based on the Entropy Time Series. The results show an excellent tradeoff between attack detection and false positive rate. This trade off can be adjusted considering the security constraints of the analyzed network. If the network requires a strict security level, a lower threshold can be set and will detect most of the threats, at a cost of a higher false positive rate. The opposite is also valid, if the network has a higher user quality constraint, a higher threshold can be set resulting in lower false positive rate at a cost of a also lower threat detection rate. Moreover, 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 measure against attackers and showing the potential to scale threat detection. This short threat detection time is achieved thanks to the stream processing technology combined with machine learning algorithms.

Regarding the Threat Prevention, we propose an architecture based on Software Defined Networking and the periodic analysis of the first five packets of each flow. The architecture ensures a fast 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 additional delay to the user communication. Moreover, the architecture 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 architecture is robust, because both the controller and the analysis machine are protected against flood attacks, since not all packets are mirrored to the analysis machine and there is a defense against spoofed IP that would flood the controller. Furthermore, the architecture has a great potential to scale, since it has a short threat detection time, due to the stream processing core that improves its throughput when the parallelism increases and since the architecture that analyzes only few packets of each flow prevents attacker to flood sensor elements.

Moreover, we propose an elastic mechanism to instantiate sensor elements according to the demand in our architecture. The controller installs rules to mirror the traffic to the sensor elements that combines the packets into flows and extract flow features to send to the Threat Detection application. We evaluate that the most critical resource for a sensor element is the processing throughput. We monitor the processing resource consumption of each sensor and detect overload scenarios, that is, when a sensor has almost 100% of CPU usage. Then the controller evaluates in which physical machine a new sensor can be instantiated and distributes the flows, according to their sources, to avoid overload in a sensor. This way, our architecture ensure that there is no loss while extracting flow features. Another possible scenario is when the traffic decreases and a sensor virtual machine can be deactivated, to avoid wasting resources. When the controller detects an unload, it manages the mirroring rules, resulting in a machine without any flow to analyze. In this case, this machine can be deactivated, releasing the resources. Therefore, the sensor elements acquire an elastic behavior, since the number of sensor is adjusted to the network usage.

7.1 Future Work

The use of the proposed threat detection architecture to detect application threats is a future work. New detection methods can be implemented to process application logs and payload data to detect application threats, such as web application attacks, in real time with the proposed architecture. Moreover, the study and implementation of more online trained threat detection techniques, such as the ones based in game theory, is also a future work. The implementation of the threat prevention scheme as a virtualized network function will also be considered as a sequence to this work.

Bibliography

[1] SUTHAHARAN, S. “Big data classification: Problems and challenges in network intrusion prediction with machine learning”, *ACM SIGMETRICS Performance Evaluation Review*, v. 41, n. 4, pp. 70–73, 2014.

[2] CLAY, P. “A modern threat response framework”, *Network Security*, , n. 4, pp. 5–10, 2015.

[3] IQBAL, S., KIAH, M. L. M., DHAGHIGHI, B., et al. “On cloud security attacks: A taxonomy and intrusion detection and prevention as a service”, *Journal of Network and Computer Applications*, v. 74, pp. 98–120, 2016.

[4] SYMANTEC. *Internet Security Threat Report.* Relatório Técnico 04, Symantec Co., 2016. accessed 12/04/17.

[5] BILGE, L., DUMITRAS, T. “Before we knew it”. In: *ACM Conference on Computer and Communications Security - CCS ’12*. ACM, 2012. doi: 10.1145/2382196.2382284.

[6] NAWROCKI, M., WÄHLISCH, M., SCHMIDT, T. C., et al. “A Survey on Honeypot Software and Data Analysis”, *arXiv preprint arXiv:1608.06249*, 2016.

[7] HU, P., LI, H., FU, H., et al. “Dynamic defense strategy against advanced persistent threat with insiders”. In: *IEEE Conference on Computer Communications (INFOCOM)*, pp. 747–755, 2015.

[8] SONG, J., TAKAKURA, H., OKABE, Y., et al. “Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation”. In: *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pp. 29–36. ACM, 2011.

[9] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., et al. “OpenFlow: enabling innovation in campus networks”, *SIGCOMM Comput. Commun. Rev., 2008*, 2008.

[10] LOBATO, A. G. P., DA ROCHA FIGUEIREDO, U., ANDREONI LOPEZ, M., et al. "Uma Arquitetura Elástica para Prevenção de Intrusão em Redes Virtuais usando Redes Definidas por Software", *Anais do XXXII SBRC 2014*, pp. 427–440, maio 2014.

[11] LOBATO, A. G. P., LOPEZ, M., DUARTE, O. C. M. B. "Um Sistema Acurado de Detecção de Ameaças em Tempo Real por Processamento de Fluxos", *XXXIV SBRC, Salvador, Bahia, Brasil*, 2016.

[12] LOBATO, A. G. P., LOPEZ, M. A., REBELLO, G., et al. "Um Sistema Adaptativo de Detecção e Reação a Ameaças", *XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg*, 2017.

[13] LOBATO, A. G. P., LOPEZ, M. A., SANZ, I., et al. "An Adaptive Real-Time Architecture for Zero-Day Threat Detection", *to appear in IEEE International Conference on Communications - ICC*, 2018.

[14] LOBATO, A. G. P., LOPEZ, M. A., DUARTE, O. C. M. B. "A Fast and Accurate Threat Detection and Prevention Architecture using Stream Processing", *to be submitted to Computer Communications*, 2018.

[15] LOPEZ, M. A., LOBATO, A. G. P., DUARTE, O. C. M. B. "A performance comparison of Open-Source stream processing platforms". In: *Global Communications Conference (GLOBECOM), 2016 IEEE*, pp. 1–6. IEEE, 2016.

[16] LOPEZ, M. A., LOBATO, A. G. P., MATTOS, D., et al. "Um Algoritmo Não Supervisionado e Rápido para Seleção de Características em Classificação de Tráfego", *SBRC'2017, Belém- Pará, PA, Brasil*, 2017.

[17] LOPEZ, M. A., SILVA, R., ALVARENGA, I., et al. "Collecting and Characterizing a Real Broadband Access Network Traffic Dataset", *1st Cyber Security in Networking Conference*, 2017.

[18] BUCZAK, A., GUVEN, E. "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection", *IEEE Communications Surveys Tutorials*, , n. 99, pp. 1–26, 2015.

[19] JI, S.-Y., JEONG, B.-K., CHOI, S., et al. "A multi-level intrusion detection method for abnormal network behaviors", *Journal of Network and Computer Applications*, v. 62, pp. 9–17, 2016.

[20] LAKHINA, A., CROVELLA, M., DIOT, C. "Mining anomalies using traffic feature distributions". In: *ACM SIGCOMM Computer Communication Review*, v. 35, pp. 217–228. ACM, 2005.

[21] FERNANDES, G., CARVALHO, L. F., RODRIGUES, J. J., et al. "Network anomaly detection using IP flows with Principal Component Analysis and Ant Colony Optimization", *Journal of Network and Computer Applications*, v. 64, pp. 1–11, 2016.

[22] PALMIERI, F., FIORE, U., CASTIGLIONE, A. "A distributed approach to network anomaly detection based on independent component analysis", *Concurrency and Computation: Practice and Experience*, v. 26, n. 5, pp. 1113–1129, 2014.

[23] RINGBERG, H., SOULE, A., REXFORD, J., et al. "Sensitivity of PCA for Traffic Anomaly Detection". In: *ACM SIGMETRICS*, pp. 109–120, 2007.

[24] AMARAL, A. A., DE SOUZA MENDES, L., ZARPELÃO, B. B., et al. "Deep IP flow inspection to detect beyond network anomalies", *Computer Communications*, v. 98, pp. 80–96, 2017.

[25] BOSTANI, H., SHEIKHAN, M. "Hybrid of anomaly-based and specification-based IDS for Internet of Things using unsupervised OPF based on MapReduce approach", *Computer Communications*, v. 98, pp. 52–71, 2017.

[26] SINGH, K., GUNTUKU, S. C., THAKUR, A., et al. "Big Data Analytics framework for Peer-to-Peer Botnet detection using Random Forests", *Information Sciences*, v. 278, pp. 488–497, 2014.

[27] RATHORE, M. M., PAUL, A., AHMAD, A., et al. "Hadoop Based Real-Time Intrusion Detection for High-Speed Networks". In: *IEEE GLOBECOM*, Washington, USA, 2016.

[28] LEE, W., STOLFO, S. J., MOK, K. W. "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*, pp. 114–124. ACM, 1999.

[29] BRINGER, M. L., CHELMECKI, C. A., FUJINOKI, H. "A survey: Recent advances and future trends in honeypot research", *International Journal of Computer Network and Information Security*, v. 4, n. 10, pp. 63, 2012.

[30] OWEZARSKI, P. “A near real-time algorithm for autonomous identification and characterization of honeypot attacks”. In: *ACM Symp. on Information, Computer and Communications Security*, pp. 531–542. ACM, 2015.

[31] BERNAILLE, L., TEIXEIRA, R., AKODKENOU, I., et al. “Traffic classification on the fly”, *ACM SIGCOMM Computer Communication Review*, v. 36, n. 2, pp. 23–26, 2006.

[32] PENG, L., YANG, B., CHEN, Y. “Effective packet number for early stage internet traffic identification”, *Neurocomputing*, v. 156, pp. 252 – 267, 2015.

[33] CHEN, Z., PENG, L., GAO, C., et al. “Flexible neural trees based early stage identification for IP traffic”, *Soft Computing*, v. 21, n. 8, pp. 2035–2046, 2017. ISSN: 1433-7479. doi: 10.1007/s00500-015-1902-3.

[34] DONATO, W. D., PESCAPE, A., DAINOTTI, A. “Traffic identification engine: an open platform for traffic classification”, *IEEE Network*, v. 28, n. 2, pp. 56–64, March 2014.

[35] SPEROTTO, A., SADRE, R., VAN VLIET, F., et al. “A labeled data set for flow-based intrusion detection”. In: *IP Operations and Management*, Springer, pp. 39–50, 2009.

[36] AMINI, M., JALILI, R., SHAHRIARI, H. R. “RT-UNNID: A practical solution to real-time network-based intrusion detection using unsupervised neural networks”, *Computers & Security*, v. 25, n. 6, 2006.

[37] SANGKATSANEE, P., WATTANAPONGSAKORN, N., CHARNSRIPINYO, C. “Practical real-time intrusion detection using machine learning approaches”, *Computer Communications*, v. 34, n. 18, pp. 2227 – 2235, 2011.

[38] JOHNSON, T., LAZOS, L. “Network anomaly detection using autonomous system flow aggregates”. In: *Global Communications Conference (GLOBECOM), 2014 IEEE*, pp. 544–550. IEEE, 2014.

[39] DU, Y., LIU, J., LIU, F., et al. “A real-time anomalies detection system based on streaming technology”. In: *Sixth International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, v. 2, pp. 275–279. IEEE, 2014.

[40] ZHAO, S., CHANDRASHEKAR, M., LEE, Y., et al. “Real-time network anomaly detection system using machine learning”. In: *11th International*

Conference on the Design of Reliable Communication Networks (DRCN), pp. 267–270. IEEE, 2015.

[41] RAI, K., DEVI, M. S. "Intrusion Detection Systems: A Review", *Journal of Network and Information Security*, v. 1, n. 2, 2013.

[42] SHIN, S., PORRAS, P., YEGNESWARAN, V., et al. "FRESCO: Modular composable security services for software-defined networks". In: *Proceedings of Network and Distributed Security Symposium*, 2013.

[43] KIM, H., GUPTA, A., SHAHBAZ, M., et al. *Simpler Network Configuration with State-Based Network Policies*. Relatório técnico, Georgia Institute of Technology, 2013.

[44] ANDREONI LOPEZ, M., DA ROCHA FIGUEIREDO, U., LOBATO, A. G. P., et al. "BroFlow: Um Sistema Eficiente de Detecção e Prevenção de Intrusão em Redes Definidas por Software", *WPerformance'2014 (XXXIV Congresso da Sociedade Brasileira de Computação - CSBC 2014)*, pp. 1919–1932, 2014.

[45] LIN, Y.-D., LIN, P.-C., YEH, C.-H., et al. "An Extended SDN Architecture for Network Function Virtualization with a Case Study on Intrusion Prevention", *IEEE Network*, v. 29, n. 3, pp. 48–53, 2015.

[46] BRAGA, R., MOTA, E., PASSITO, A. "Lightweight DDoS flooding attack detection using NOX/OpenFlow". In: *IEEE 35th Conference on Local Computer Networks*, pp. 408–415, 2010.

[47] RYCHLY, M., KODA, P., SMRZ, P. "Scheduling Decisions in Stream Processing on Heterogeneous Clusters". In: *Eighth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, pp. 614–619, 2014.

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

[49] CHENG, Z., CAVERLEE, J., LEE, K. "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*, CIKM'10, pp. 759–768. ACM, 2010. ISBN: 978-1-4503-0099-5.

[50] LIPPMANN, R. P., FRIED, D. J., GRAF, I., 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.*, v. 2. IEEE, 2000.

[51] TAVALLAEE, M., BAGHERI, E., LU, W., et al. "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.

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

[53] GARCIA, S., GRILL, M., STIBOREK, J., et al. "An empirical comparison of botnet detection methods", *Computers & Security*, v. 45, pp. 100–123, 2014.

[54] MORAES, I. M., MATTOS, D. M., FERRAZ, L. H. G., et al. "FITS: A flexible virtual network testbed architecture", *Computer Networks*, v. 63, n. 0, pp. 221 – 237, 2014. ISSN: 1389-1286. doi: http://dx.doi.org/10.1016/j.bjp.2014.01.002. Special issue on Future Internet Testbeds Part {II}.

[55] BEZERRA, G. M. G., MATTOS, D. M. F., FERRAZ, L. H. G., et al. "Sistema automatizado de gerência de recursos para ambientes virtualizados", *publicado em XXXII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos-SBRC*, 2014.