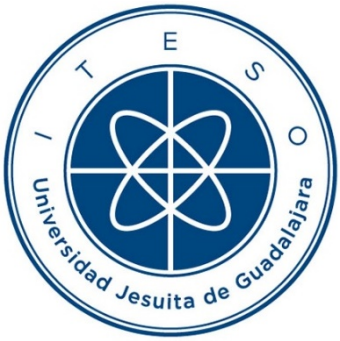
INSTITUTO TECNOLÓGICO
Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Doctorado en Ciencias de la Ingeniería



SISTEMA DE DETECCIÓN DE INTRUSOS BASADO EN ANOMALÍAS A NIVEL DE HOST UTILIZANDO RANKINGS PARA LA SEGURIDAD INFORMÁTICA

Tesis que para obtener el grado de
Doctor en Ciencias de la Ingeniería
presenta: Alvaro Iván Parres-Peredo

Director de tesis: Dr. Hugo Ivan Piza Davila
Co-director de tesis: Dr. José Francisco Cervantes Alvarez

Tlaquepaque, Jalisco. Marzo de 2020

TITULO: **Sistema de Detección de Intrusos basado en Anomalías a nivel de Host utilizando Rankings para la Seguridad Informática**

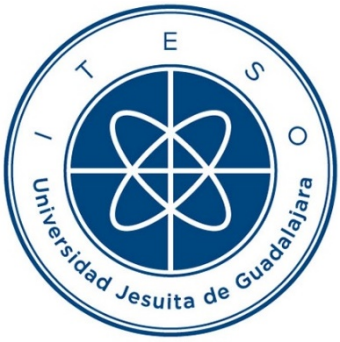
AUTOR: Alvaro Iván Parres-Peredo
Ingeniero en Sistemas Computacionales (ITESO, México)
Maestro en Administración de Tecnologias de Información (ITESM, México)

DIRECTOR DE TESIS: Hugo Ivan Piza Davila
Departamento de Electrónica, Sistemas e Informática, ITESO
Ingenierio en Sistemas Computacionales (Instituto Tecnológico de Colima)
Maestro en Ciencias de la Eléctrica (Cinvestav, México)
Doctor en Ciencias de la Eléctrica (Cinvestav, México)

NÚMERO DE PÁGINAS: XXVII, 128

ITESO – The Jesuit University of Guadalajara

Department of Electronics, Systems and Informatics
Doctoral Program in Engineering Sciences



HOST-LEVEL ANOMALY-BASED INTRUSION DETECTION SYSTEM USING RANKINGS FOR CYBERSECURITY

Thesis to obtain the degree of
Doctor in Engineering Sciences
Presents: Alvaro Iván Parres-Peredo

Thesis Director: Dr. Hugo Ivan Piza Davila
Thesis Co-director: Dr. José Francisco Cervantes Alvarez

Tlaquepaque, Jalisco, Mexico

March 2020

TITLE: **Host-Level Anomaly-Based Intrusion Detection System using Rankings for Cybersecurity**

AUTHOR: Alvaro Iván Parres-Peredo
Bachelor's degree in Computer Systems (ITESO, México)
Master's degree in administration of information technology (ITESM, México)

THESIS DIRECTOR: Hugo Ivan Piza Davila
Departament of Electronics, Systems, and Informatics, ITESO
Bachelor's degree in Computer System Engineering (Instituto Tecnologico de Colima)
Master's degree in electrical engineering (Cinvestav, Mexico)
Ph.D degree in electrical engineering (Cinvestav, Mexico)

NUMBER OF PAGES: XXVII, 128

To her,

the woman who without hesitation said yes

to this adventure, her name Jeannifer,

my life partner, my wife.

To them,

the origin of the sound that never stuns

that becomes the source of my strength,
the laughter of my sons Luca and Marcelo.

Resumen

En los sistemas computacionales y las redes informáticas, la seguridad es un área de investigación en constante evolución. Desde que Anderson propuso un sistema de detección de intrusos, muchos investigadores han dirigido sus trabajos hacia este tipo de sistemas con el objetivo de detectar ataques conocidos y desconocidos con la mayor precisión posible.

Este trabajo comienza presentando una descripción de los sistemas de detección de intrusos y sus desafíos en el campo de la seguridad de las redes de computadoras. Enseguida, presenta una revisión de trabajos de investigación pertenecientes al *State-of-the-art* sobre sistemas de detección de intrusos basados en anomalías, que están destinados a detectar nuevos tipos de ataques.

Los sistemas de detección de intrusos basados en anomalías utilizan perfiles para caracterizar el comportamiento esperado de los usuarios de la red. La mayoría de estos sistemas construye un sólo perfil que caracteriza a todo el tráfico de la red.

Este trabajo propone una metodología de detección de intrusos basada en anomalías a nivel usuario que utiliza solo el tráfico en el *host*. El perfil propuesto es una colección de *TopKs* de los servicios alcanzados por el usuario. Para detectar los comportamientos inesperados, el tráfico en tiempo real también se organiza en *TopKs* y se compara con el perfil utilizando medidas de similitud. Todas las medidas de similitud se procesan utilizando un filtro de promedio deslizante para calcular el comportamiento predominante. Este valor se utiliza para determinar si el usuario está exhibiendo o no un comportamiento esperado en un momento dado.

Los experimentos demostraron que la metodología propuesta fue capaz de detectar un tipo particular de ataque de *malware* para todos los usuarios probados.

Summary

In computer systems and computer networks, security is a research area in constant evolution. Ever since Anderson proposed an intrusion detection system, many researchers have led their works towards that area with the aim of detecting both known and unknown attacks with the highest precision.

This work starts with a general overview of Intrusion Detection Systems as well as their challenges in computer network security field. Consequently, it presents a review of state-of-the-art research works on anomaly-based intrusion detection systems, which are intended to detect new types of attacks.

Anomaly-based intrusion detection systems use profiles to characterize expected behavior of network users. Most of these systems build a single profile that characterizes the entire network traffic.

This work proposes a user-level anomaly-based intrusion detection methodology using only the network traffic at the host. The proposed profile is a collection of TopK rankings of reached services by the user. To detect unexpected behaviors, the real-time traffic is organized into TopK rankings and compared to the profile using similarity measures. All the similarity measures are processed by means of a moving-average filter which calculates a predominant behavior. This value is used to determine whether the user is having or not an expected behavior.

The experiments demonstrated that the proposed methodology was capable of detecting a particular kind of malware attack for all the users tested.

Acknowledgements

The author wishes to express his sincere appreciation to Dr. Hugo Ivan Piza Davila, professor of the Department of Electronics, Systems, and Informatics at ITESO, for his encouragement, expert guidance and keen supervision as doctoral thesis director throughout the course of this work. Also his gratitude to Dr. José Francisco Cervantes Alvarez, from ITESO, for his support as doctoral thesis co-director during the development of this work. He also thanks Dr. Guillermo Sanchez Días, Dr. Iván Esteban Villalon Turrubiates, and Dr. Dr. Luis Julián Dominguez Perez, members of his Ph.D. Thesis Committee, for their interest, assessment, and suggestions.

The author express his gratitude to Dr. Victor Hugo Zaldivar Carrillo who in a first moment, as academic coordinator, encouraged the author to show interest on taking a Ph.D., and then as Director of the Department of Electronics, Systems, and Informatics at ITESO, he provided the author all the support and resources required to accomplish his Ph.D.

It is the author's pleasure to acknowledge the collaboration, discussions and advice of all his colleagues in the Department of Electronics, Systems, and Informatics at ITESO, specially to Dr. Omar Longoria, Dr. Luis Rizo, and Dr. Roberto Osorno.

The author gratefully acknowledges the authorities of ITESO for the financial support through an assistantship granted by the ITESO's Program for Academic Level Enhancement (Programa de Superación del Nivel Académico, PSNA).

Also, the author wants to express his appreciation to his mother Dr. Alicia Peredo Merlo and Dr. Miguel Angel Navarro Navarro for their constant encouragement to finish the Ph.D. program successfully, as well as to Armando Allende de la Torre and Jeanette Pantoja Yuen for all their support words along the Ph.D. program.

The most special thanks are to my family: my sons, Leon Marcelo and Luca Mariano, who, with their laughter, were the source of the courage to accomplish this work, and my wife, Jeannifer Allende Pantoja, for whom I have the most important and sincerely appreciation for her understanding and unconditional support during this common project.

And, finally I want to sincerely apologize to my children and wife for each one of the lost moments due to my Ph.D. studies.

Contenido

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

The Internet has been growing at a very high rate, becoming the primary global media. Due to the development of novel computing technologies and the "As Service" model, the Internet has also become the operations center of many organizations. At present, a wide variety of data is traveling on the Internet: from simple email to the entire operations data of a company. This makes computer network security more critical than ever.

Day after day, information systems suffer from new kinds of attacks. As these attacks become increasingly complex, the technical skill required to create them is decreasing[58].

The term computer security is defined by the National Institute of Standards and Technology(NIST) [35] as follows: the protection afforded to an automated information system in order to attain the applicable objectives of preserving the integrity, availability, and confidentiality of information system resources (including hardware, software, firmware, information/data, and telecommunications).

Migga[44] defines computer security as a branch of computer science that focuses on creating secure environments for the use of computers. It focuses on the behavior of the users of computers and related technologies, as well as on the protocols required to create a secure environment for everyone. When we talk about computer network security, the secure environment involves all network resources: computer, data, devices, and users.

At present, firewalls and access control systems are no longer enough to protect computer systems. Intruders find new ways to attack computers and systems. This motivated the rise of a new layer of security called the intrusion detection system (IDS). The first approach of an IDS was proposed by Anderson[13] in 1980. An IDS intends to identify intruders (or attackers) by monitoring and analyzing the events on systems, computers and/or networks. Figure 1 shows the security methods on a simple computer network diagram.

Current IDSs are classified according to the approach employed to detect intrusions. The most popular approaches are: signature based and anomaly based. The former is very efficient in detecting well-known attacks, but it is quite inefficient in detecting new forms of attacks. The latter is more

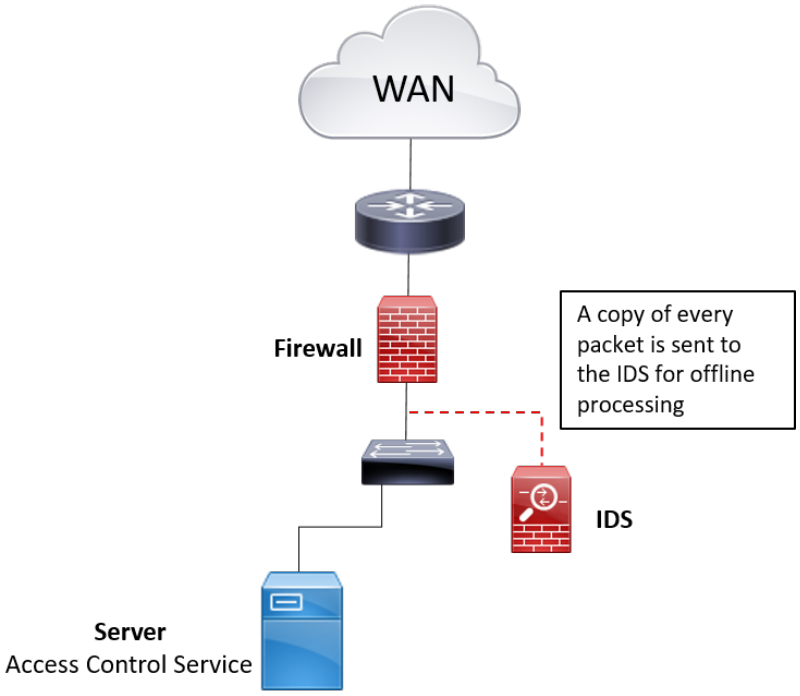


Figure 1: Diagram of a simple network connected to a Wide Area Network (WAN) with a firewall and an intrusion detection system (IDS).

efficient in detecting new forms of attacks, but it has high rates of false positives.

On the other hand, traditional authors like Stalling[72] define three types of network based on the geographical scope: (1) local area networks (LANs), (2) metropolitan area networks, and (3) wide area networks. Modern authors like Edwards Wade [27] incorporate new types of network like the campus area network (CAN), which is defined as a group of LAN segments interconnected within a building or group of buildings that form one network. Typically, the company owns the entire network, including the wiring between buildings, in contrast to metropolitan area networks.

In large organizations such as universities, many users (students, employees, visitors) are connected to the campus area network (CAN) to either access intranet services or obtain Internet access, from different kinds of devices. The probability for a network attack to be originated from inside the CAN is high for two main reasons: (a) malicious behaviors of inexperienced users practicing some hacking technologies (script kiddies), and (b) privileged users are the victim of social engineering attacks when clicking links on e-mails or web pages from untrusted sources.

We believe that a viable way to prevent these security problems is by detecting when a user is

having abnormal network behavior. This involves building individual network profiles representing the normal behaviors of every user in an organization. To do this, real-time traffic has to be captured from the nearest point to each user's access device or even from the same device.

In this work, we propose a methodology capable of detecting when a network user is having an abnormal behavior and, therefore, could be the victim of a network attack.

Our proposal uses network traffic captured at the host machine. We build a TopK ranking containing the services with the highest amount of bytes transferred from/to the host during a time-frame. Using TopK rankings for user profiling is a novel element in the design of anomaly-based IDS.

Most of the state-of-the-art anomaly-based IDSs use the traffic captured at the border of the network. Thus, their profiles represent the behavior of the entire network segment. In contrast, our profile reproduces the behavior of a single user. Even though our proposal is clearly less scalable, our focus is on protecting privileged users in the organization, who execute critical tasks, from internal and external threats.

This doctoral dissertation is organized as follows:

Chapter 1 presents an overview of Intrusion Detection Systems and explains each one of the detection methodologies defined by that the NIST: signature-based, anomaly-based and stateful protocol analysis.

Chapter 2 introduces a classification of network security attacks, and describes two main threats present at campus area networks.

Chapter 3 is a review of the state of the art on Anomaly-Based Intrusion Detection Systems, organized by scope: network, host and hypervisor.

Chapter 4 presents the first proposal of a user profile which is built from the host network traffic, and is organized in four views: 1) IP View, 2) Services View, 3) HTTP Host View, and 4) HTTP URL View.

Chapter 5 proposes to define the user profile as a collection of TopK rankings for each view. Each TopK contains the k elements with the highest amount of bytes transferred during a time frame.

Chapter 6 presents a couple of experiments where the previously proposed profile is used. The first experiment compares real-time traffic with the user profile, and the second identifies the owner of a given real-time traffic.

Chapter 7 describes and analyzes the time and space complexity of the algorithm that computes the set of TopK rankings that built up a user profile.

Chapter 8 proposes a MapReduce implementation for both the profiling process and the analysis of real-time traffic. The purpose of this implementation is to accelerate the run time of the algorithms by workload distribution.

Chapter 9 presents a full-serverless architecture and implementation for both the profiling process and the analysis of real-time traffic. The purpose of this implementation is to reduce the costs originated with the MapReduce implementation.

Chapter 10 proposes a methodology to compute the predominant behavior of a network user, in terms of: current real-time network and the profile.

Chapter 11 presents some experiments that we carried out to select the parameters that best determine whether a user is having or not an unexpected behavior.

Finally, this thesis includes two appendix. Appendix A shows the reference list of the thirteen internal research reports that I wrote during the doctoral studies, and Appendix B shows the list of conference and journal papers published during my doctoral studies.

1. Intrusion Detection Systems

Intrusion detection is the process of monitoring the events occurring within a computer system or network, and analyzing them to search for signs of possible violation of computer security policies. These events might have a malicious nature, such as malware or attackers. An is a software system that automates the intrusion detection process [68].

NIST defines three detection methodologies: (1) signature-based detection; (2) anomaly-based detection; and (3) stateful protocol analysis[68].

1.1. Signature-Based Detection

This detection technique is the first one employed on an and was introduced in Anderson reports[13]. The intrusion is detected by matching the behavior recorded in either log records, network packets, or system status against well-known suspicious patterns. This methodology is very effective in detecting known attacks, but is useless for new forms of attacks[25].

Examples of malicious behaviors and related attacks that this methodology is able to detect are described in Table 1.1, where we can observe that the expected behaviors are well defined. Figure 1.1 depicts this detection methodology in a diagram.

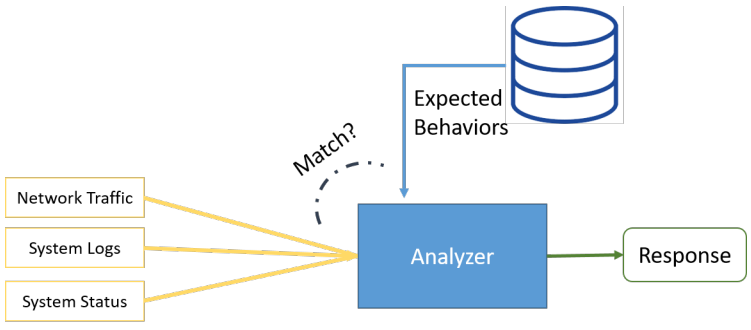


Figure 1.1: Diagram of a generic signature-based IDS.

TABLE 1.1. KNOWN ATTACKS AND THEIR EXPECTED BEHAVIOR.

Attack	Expected Behavior
Unauthorized access	SSH try of a root login
Unauthorized web access	Multiple wrong password web login forms submitted from the same IP address in a short period of time
Malware execution	High CPU load into a server and multiple outgoing TCP connections
Social Engineering and malware	An email with subject "Urgent Document" and an attachment filename "authorization.exe"

1.2. Anomaly-Based Detection

Anomaly-based detection is the process of comparing definitions of what is considered normal (i.e., profiles) against observed events in order to identify significant deviations. The profiles are built by monitoring the characteristics of typical activity over a period of time[68]. An advantage of this methodology is its ability to find new forms of attacks, but normally it has high rates of false positives. Figure 1.2 depicts a generic diagram of this detection methodology.

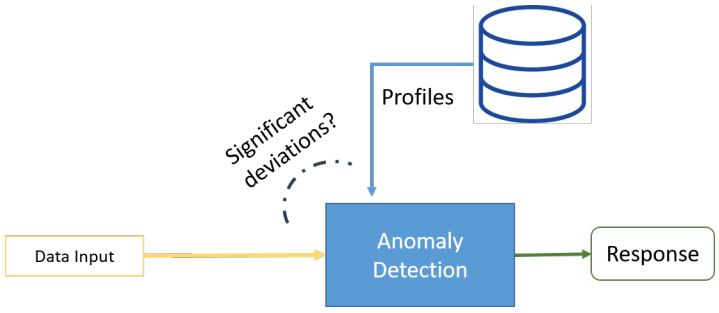


Figure 1.2: Diagram of a generic anomaly-based IDS.

Anomaly-based IDS is considered to be one of the foremost research areas in network security[50]. The detection methods and the selection of the system or network features to be monitored are two open issues[25].

Many research works have been developed around the classification of network traffic using different machine-learning techniques, with the aim of achieving 100% intrusion detection and a low rate of false positives. Two classes of machine learning techniques are used: (a) single and (b) hybrid classifiers. Examples of single machine-learning classifiers include the following: support vector machines, self-organizing maps, neural networks, and k-nearest neighbors. Hybrid machine-learning classifiers have the purpose of acquiring a superior probable accuracy for intrusion detection; these classifiers combine several machine-learning techniques to improve their performance[50].

1.3. Stateful Protocol Analysis

Stateful protocol analysis or deep packet inspection is the process of comparing predefined accepted protocol activity against the observed events to identify deviations. Unlike anomaly-based detection, which uses network profiles, stateful protocol analysis relies on vendor specifications of how the protocol should and should not be used[68]. In Figure 1.3 we can observe a generic diagram of this type of detection methodology.

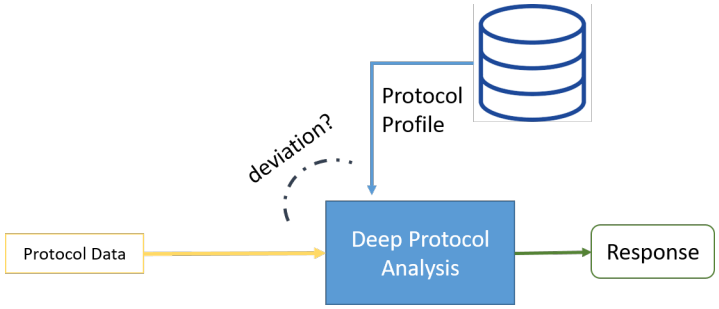


Figure 1.3: Diagram of a generic stateful-protocol IDS.

Usually, the vendor specifications include: rules for individual commands, sequence of commands, minimum and maximum lengths for arguments, argument data types, etc.

This methodology produces a low false-positive rate similar to signature-based detection because it is based on legitimate behaviors. In addition, it is able to detect unexpected behaviors[47]. The big challenge of this methodology is to design secure protocol specifications strong enough to detect illegitimate behaviors.

2. Challenges and Opportunities in Computer Network Security

2.1. Computer Network Attacks

Computer networks attacks can be classified according to different criteria; for instance, based on their objectives, types of target, or the way they are executed.

Stallings[71] classifies security attacks into two big categories: passive attacks and active attacks. A passive attack attempts to learn or make use of information from the system but does not affect system resources. An active attack attempts to alter system resources or affect their operations.

Kandall[42] proposes a classification which is widely used in research works. He defined the following classes:

a) Denial of Service (DOS): is an attack in which the attacker makes some computing or memory resource too busy or too full to handle legitimate requests, or denies legitimate users access to a machine.

b) User to Root: is a type of "exploit" program in which the attacker starts out with access to a normal user account on the system and is able to exploit some vulnerability in order to gain root access to the system.

c) Remote to User: occurs when an attacker with the ability to send packets to a machine over a network but not having an account on that machine, exploits some vulnerability to gain local access as a user of that machine.

d) Probe: refers to a program that can automatically scan a network of computers to gather information or to find known vulnerabilities.

On the other hand, Heerden[37] provides other taxonomy of network attacks that he calls attack scenarios, and includes the following: denial of service, industrial espionage, web deface, spear phishing, password harvesting, snooping for secrets, financial theft, amassing computer resources, industrial sabotage and cyber warfare.

2.2. Challenges in Computer Network Security

In order to visualize the opportunity areas on computer network security, we distinguish border security from internal security. The former focuses on protecting network components from external attacks, i.e., those from outside networks. The latter focuses on protecting devices from attacks performed from another device on the same network.

Most of computer network research has focused on border security, more specifically, on detecting denial of service attacks or unprivileged access, by analyzing the network traffic. However, there are currently many issues and threats on internal security not yet been studied by current research. We identify two types of such threats as follows: 1) privileged users at networks and 2) script kiddies at campus area networks.

2.2.1 Privileged Users

The 2015 annual CISCO Security report[1] identifies as a key discovery that both users and their equipment have become unwitting parts of security problems. Some examples that CISCO gives about this are:

a) Online criminals rely on users to install malware or help exploit security gaps.

b) Malware creators are using web browser add-ons as a medium for distributing malware and unwanted applications. This approach to malware distribution has proved to be successful for a malicious actor because many users inherently trust add-ons or simply view them as benign.

c) User's careless behavior when using the Internet, combined with targeted campaigns by adversaries, place many industry verticals at higher risk of web malware exposure

One vulnerable element for a system is a user that has rights over it. One challenge of computer network security is to prevent privileged users from being accomplices of attackers that employs users' computers to gain access to information systems and/or network services.

2.2.2 Script Kiddies

Nowadays, there exists many tools, software applications, and scripts capable to perform penetration testing on information systems and computer networks, with the sole intention of attacking them. Kali Linux , a Linux distribution, includes over 600 pre-configured penetration-testing tools, which can be treated as penetration test tools, or forms of attack.

We define script kiddie as a user that thinks of himself as a hacker but has very low technical skills. He/she does not write his/her own code; instead, he/she runs scripts written by more skilled attackers[58].

Script kiddies inside a considerably large network, such as a CAN, represent a serious security risk; their own amateurism, poor knowledge about what they are doing, and few technical skills might result in the following scenarios:

a) They might open a backdoor for a professional attacker. Because of the download and execution of programs and scripts that comes from unknown sources, the script kiddie can download a malware that can generate a back door.

b) The incorrect execution of some types of attacks, like man in the middle, can provoke a downtime on the network for an unknown period of time that also generates a lower service level.

c) On public networks, like coffee shop's hotspot, the presence of script kiddies can generate an uncomfortable experience to regular network users.

The challenge of computer network security is to identify script-kiddie behaviors in the computer network of an organization, such that the corresponding risk can be prevented.

2.3. Conclusions

In this chapter, we presented how the computer network security is becoming more important and critical for organizations, because of the high value of the information that travels across the network, and the costs of IT systems downtime.

The computer network attacks are been studied for a long time and new technologies for preventing them are being developed. Most research works focus on designing or improving intrusion

detection systems in order to detect more kind of attacks than a simple firewall, or access control system.

The intrusion detections systems have been employed also to detect new patterns of attacks using machine-learning algorithms, each time more efficient and precise. Most of these systems work at the border of the network or at some specific points, regularly, between the datacenter and the network. Since many attacks come from inside the organization network, we have identified opportunities for improving network security by profiling end-users according to their normal network behavior and monitoring their devices.

3. Review of the state of the art on Anomaly-Based Intrusion Detection Systems

The following sections present an overview of different research works in the state of the art on anomaly-based IDS, organized by scope.

3.1. Network Anomaly-based Intrusion Detection Systems

Network Anomaly-based Intrusion Detection Systems (NAIDS) monitor network traffic for particular network segments and analyze network, transport, and application protocols to identify suspicious activity [68].

NAIDS regularly collect data to be analyzed from a gateway at the border of the network; thus, they identify attacks or anomalies from the entire network segment.

In accordance to NIST[68], an Anomaly-based IDS identifies attacks by calculating deviations between real-time traffic data and a profile. However, there are many research works that identify attacks by calculating the resemblance between traffic data and a known pattern, and they are still self-considered anomaly-based detection methodologies.

We organized the overview of NAIDS research works in three categories, according to the underlying technique that detects attacks: 1) Machine Learning algorithms, 2) Deep Learning algorithms, and 3) Statistics functions. The two first categories regularly works with pre-built datasets witch are used to train and validate the models. The third is more aligned to the NIST definition where a profile is built to compare or evaluate traffic. Tables 3.1,3.2, and 3.3 present some of these works on each one of the categories, respectively.

TABLE 3.1. MACHINE LEARNING-BASED NA-IDS

Article	ML Algorithm	Dataset	Results	Observations
[66]	• Random Forest • Naive Bayes • Extreme Gradient Boosting • Support Vector Machine	CIC-IDS2018[21]	They conclude that the Extreme Gradient Boosting and Random Forest performs better than the others using the following respective recalls: 99.87% and 99.75%	This paper proposed to use these two algorithms for feature selection: Boruta Feature Selection, and Recursive Feature Elimination
[36]	Author proposed: Multimodal-Sequential Approach with Deep Hierarchical Progressive Network	• NSL-KDD[23] • UNSQ-NB15[54] • CIC-IDS2017[22]	Accuracy, precision and recall of 0.99 on CIC-IDS2017 dataset	The authors grouped the features in three categories: 1) packet, 2) traffic and 3) general
[63]	Author proposed: Logistic Regression LR, Decision Tree, Random Forest, Multilayer Perceptron, Naive Bayes on Apache Spark	MAWILab[29]	Best Algorithm: Logistic Regression with a Accuracy of: 96%	

TABLE 3.1. MACHINE LEARNING-BASED NA-IDS

Article	ML Algorithm	Dataset	Results	Observations
[39]	Stacked ensemble learning: Autoencoder, Support Vector Machine, and Random Forest models	• NSL-KDD[23] • UNSQ-NB15[54]	Accuracy, Precision and Recall of 0.91, 0.93 and 0.91 on CIC-IDS2017 dataset. Accuracy, Precision and Recall of 0.96, 0.94 and 0.97 on Real Traffic.	The implementation was tested on a widely used dataset and then implemented on real traffic.
[6]	Fully Bayesian-based approach for infinite bounded Generalized Gaussian mixture model	• KDD-CUP99[38] • Kyoto 2006+[46] • ISCX[69]	Average Accuracy over 86% in all cases	

TABLE 3.2. DEEP LEARNING-BASED NA-IDS

Paper	DL Algorithm	Dataset	Results	Observations
[48]	Convolutional Neural Network CNN and Long Short-Term Memory Neural Network	• CIC-IDS2017[22] • CTU-13 [31]	98.90% of Accuracy and 90.13 of Detection Rate%	The authors inject traffic data from both datasets in MAWI network traffic
[18]	Deep Neural Networks	Raw Traffic from MAWILab[29]	Detection rate: 89.4%, accuracy: 87.1%	The authors work with raw traffic in real time.

TABLE 3.2. DEEP LEARNING-BASED NA-IDS

Paper	**DL Algorithm**	**Dataset**	**Results**	**Observations**
[67]	Cycle-Generative Adversarial Networks and Multilayer Perceptron	ADFA-LD[20]	Increase the detection of unseen anomalies from 17% to 80%	The authors transform normal data into anomalous data and add this to the dataset to train the MLP.
[83]	Bi-directional long short-term memory Neural Network	UNSW-NB15[54]	High performance on identifying the presence of an attack. Weak performance on identifying the type of the attack	
[7]	Long Short-Term Memory-Recurrent Neural Network	CIDDS-001 [65]	Accuracy: 84.83%	
[55]	• Deep Convolutional Neural Network • Autoencoders • Long Short Term Memory Recurrent Neural Network	NSL-KDD[23]	Algorithms with best performance: DCNN (accuracy: 85%) and LSTM (accuracy: 89%)	

TABLE 3.2. DEEP LEARNING-BASED NA-IDS

Paper	DL Algorithm	Dataset	Results	Observations
[3]	• Deep Neural Network • Random Forest • Vote Classifier • Variational Autoencoder	CIDDS-001[65]	Accuracy over 98%	Handles imbalanced datasets with respect to the number of attack and normal events

TABLE 3.3. STATISTICAL NA-IDS

Paper	Method	Dataset or Traffic	Results	Observations
[19]	An adaptive threshold method to classify traffic in three categories: normal, suspicious and malicious.	Real traffic is collected at University of Rhode Island	Successfully detects a host-scan attack pattern against a node and/or a possible malicious node	
[30]	Sparse Coding and Frequency Domain	CAIDA[75]	Accuracy: 99.11%	

Most of these works rely on fine-tuning a particular machine-learning model until it best classifies a training set containing only artificial traffic data. Their focus is the design of a classifier instead of the discovery of new types of attacks. Only a few research works [18, 63] use real traffic data and,

therefore, have an actual chance to identify new types of attacks.

3.2. Host Anomaly-based Intrusion Detection System

Host Anomaly-based Intrusion Detection Systems (HAIDS) monitor the characteristics of a single host and the events occurring within that host for suspicious activities. The sources of data for HAIDS may include the following: wired and wireless network traffic from/to the host, system logs, running processes, file access and modifications, and system/application configuration changes.[68]

A common challenge that HAIDS deal with is the lack of available datasets containing events either from a single host or classified by host. Nevavuori and Kokkonen [56] present the requirements and challenges to develop a host dataset for intrusion detection systems.

Most of the HAIDS use system events as input, for example: system calls, CPU usage, memory usage and/or network usage. They do not consider network traffic data. Some examples of these works include the following: [12, 41, 49, 74]. The following research works do consider network traffic as input data for intrusion detection: [14, 17, 61, 76, 77].

3.3. Hypervisor Anomaly-based Intrusion Detection System

Hypervisor Anomaly-based Intrusion Detection Systems (HyAIDS) monitor and analyze communications between virtual machines (VM), the hypervisor and the virtual network manager [53]. One of the benefits of the HyAIDS is the huge amount of available data due to the control that an hypervisor exerts over the running VMs and the virtual network manager.

Dildar [26] proposes a traditional architecture for HyAIDS that monitors the hypervisor events, whereas Xing [81] proposes a modern auto-scale architecture based on SDN and Containers.

In [2, 5, 57], the authors compare real-time events to a given profile with the aim of identifying deviations. Other research works like [45] propose to build HyAIDS using machine learning algorithms.

3.4. Conclusions

In computer networks, the capability to identify intrusions, that might be attacks, is an open challenge. Each new proposal aims at detecting known or unknown attacks with the highest precision

rate.

After reviewing state-of-the-art research works on anomaly-based IDS, it is clearly to see that most of them works at the network segment scope. Besides, most of them adapts a technique based on either machine learning or deep learning to detect anomalies. Some of these works validate their proposals using real-time network traffic, whereas others still use synthetic datasets for both training and testing the models. Very few research works located at the network segment scope use statistical functions to identify anomalies. All these works operate with real-time traffic and are closely aligned to an Anomaly-based IDS as defined by NIST.

We have identified two types of that work at the host that differ from each other on the source of data used to detect intrusion, namely: 1) system calls, and 2) network traffic. In both cases, real-time data is used for training and testing the models. The that work at the hypervisor scope uses the same data sources.

Finally, we can conclude that the intrusion-detection problem has been very attractive for evaluating novel machine-learning techniques. However, they are not closely aligned to the NIST definition of an anomaly-based IDS because they focus on detecting an attack instead of calculating the deviation with respect to the expected network behavior. In contrast, most of non machine-learning techniques do not use patterns of known attacks to train their models; thus, they do not generate dependencies to well-known attacks.

4. A Novel User Network Profile Based on Host Network Traffic

Intrusion detections systems have been employed to detect new patterns of attacks using machine-learning algorithms. Most of these systems work at the border of the network, assuming all the attacks come from outside.

In large organizations, like universities, many users (students, employees, visitors) are connected to the campus area network for accessing intranet services or getting internet access, through different kinds of devices. The probability for an attack to occur from inside is high because of two main reasons: a) malicious behaviors of inexperienced users when putting in practice some hacking technologies, e.g., the script kiddies, and b) privileged users being careless when clicking links at e-mails and web pages from untrusted sources or being victims of social engineering attacks.

We think that a viable way to prevent these security problems is by detecting when a user is having an abnormal network behavior or his/her behavior is similar to a well-known malicious behavior like the one of a script-kiddie. This involves building individual network profiles representing normal behaviors of every end user of the organization. To do this, real-time traffic has to be captured from the nearest point to each user access device.

In this chapter, we propose a 4-view network profile constructed from a dataset of 11 fields. The proposed dataset is built with real network traffic.

The frames are captured at user devices but they can be captured at either the network access switch, the wireless access point or the host's default gateway, depending on the network topology and configuration.

4.1. Related Work

Many research works about intrusion detection systems and computer network security validate their proposals using common datasets like KDD-CUP99[42], which is an artificial dataset for testing intrusion detection systems, and NLS-KDD[69], built at the University of New Brunswick providing a more realistic scenario since the traffic is generated by agents based on real profiles[69].

The main problem of these datasets is that they have many fields that are calculated or specific to an application, e.g. "number of failed logins", such that they are not available or easy to calculate on raw real network traffic.

The few authors that work with real network traffic capture it at the border of the networks, typically at the gateway or firewall[16], or at internet backbone link[82].

Badea[16] says that for a network administrator it is very important to understand the user behavior in computer networks. The user behavior is defined by the analysis of logged events, and an event is defined by protocols and ports. Badea[16] proposes a system for detecting the abnormal behavior of users as a Security Information and Event Management (SIEM). It is a combination of two separate legacy products: Security Information Management (SIM) and Security Event Manager (SEM). The former provides long term storage, analysis and reporting of recorded data; the latter deals with real-time monitoring, event correlation, notification and the possibility of supervision from the console.

The implementation of Badea[16] collects the events from a firewall where the packet is checked by the firewall rules and, after the blocking decision, sent to the abnormal user detection system which is an implementation of OSSIM that can collect, normalize and correlate security events occurring within a local network.

Kuai[82] proposes a different approach for profiling traffic behavior: he identifies and analyzes cluster of hosts or applications that exhibit similar communication patterns. In this approach, he uses bipartite graphs to model network traffic at the internet-facing links of the border router; then, he constructs one-mode projections of bipartite graphs to connect source hosts that communicate with the same destination host(s) and to connect destination hosts that communicate with the same source host(s). This one-mode projection graphs enable to build similarity matrices of internet end-host, with similarity being characterized by the shared number of destinations or sources between two hosts. Based on these end-hosts matrices, at the same network prefixes, a simple spectral clustering algorithm is applied to discover the inherent end-host behavior cluster.

Kuai[82] carries out an analysis over a 200 GB dataset collected from an internet backbone of 8.6 GB/s bandwidth. The data was reduced by adding packet traces into 5-tuple network flows. The dataset was built using 24-bit network prefixes with timescale of 10 s, 30 s and 1 min; these timescales were chosen because they produced the highest percentages of hosts in the top cluster.

Kuai concluded the following: 1) there is no correlation between the number of observed hosts and the number of behavior clusters, 2) the majority of end-hosts stay in the same behavior cluster over time, and 3) the profiling of network traffic in network prefixes detect anomalous traffic behaviors.

A similar approach was employed by Qin[64] but working only with traffic at port 80 (HTTP protocol), and integrating the destination URL, not only the IP address. One of the conclusions is that 93% of the hosts remain on the same behavior cluster.

However, not all the security problems occur at the network border, they also occur internally, e.g., 1) Arp Spoof Attack, 2) DNS Spoofing, 3) ICMP Redirect Attack, and 4) Wireless Replay Attack.

4.2. 4-View Network User Profile

In order to identify security issues, we propose a methodology for profiling normal user behavior, so that any network trace not following this profile can be considered as suspicious. The profile proposed is built upon the network traffic that the user generates, and is organized by the following levels of abstractions or views, each one having a specific security purpose, additional to the purpose of making the user profile:

a) Remote hosts communications or IP view.

b) Remote services accessed or service view.

c) Web hosts visited, and or host view.

d) HTTP URLs accessed or URL view.

4.2.1 Remote Host Communications or IP View

The main purpose of this view is to identify communications with remote devices which do not belong to the user profile. The IP addresses of all the remote hosts that the user established connection with during a period of time is obtained. For each host, the total amount of incoming and outgoing bytes are calculated. This view includes only the top IP addresses according to the sum of incoming and outgoing amount of bytes transferred. The number of packets is ignored because of the uncontrolled packet segmentation allowed by IPv4.

4.2.2 Remote Services Accessed or Service View

Helping to detect Trojans or malware installed at the user equipment is the main purpose of this view, which presents the top network services that the user reaches according to the sum of incoming and outgoing amount of bytes transferred. In this view, a network service is defined as a 3-tuple: <remote IP address, transport protocol, remote port>.

This view also helps to detect unauthorized services installed at internal hosts or servers. Most of the research works on intrusion detection consider this view, but located at the border of the organization network.

4.2.3 Web Hosts Visited or Host View

This view includes the top HTTP hosts visited by the user according to the number of times each host has been requested. This view is included in the user profile because of two main reasons: 1) since some networks use a HTTP Proxy, all the HTTP packets would have the same remote IP address; 2) nowadays, the HTTP servers support multiple domains and websites at the same IP address. These two reasons make the first two views less precise.

4.2.4 HTTP URLs Accessed or URL View

The last view provides the top visited URLs and HTTP methods employed according to the number of requests performed. This view has the aim of detecting JavaScript-based attacks that occur as a consequence of auto-execution of JavaScript programs by web browsers. The Cross-Site Scripting (XSS) attack[34] is an example of this. The BeEF project is a penetrating testing framework that focuses on web browsers providing tools to develop this kind of attacks.

4.3. Dataset Required for Network User Profile

In order to generate the four views proposed in the previous section, it is necessary to extract some specific data from the network traffic structured on the basis of the TCP/IP model.

4.3.1 Network Layer Model

The network traffic is comprised of packets; each packet encapsulates data organized in layers in accordance with the TCP/IP Network Model[72]. Fig. 4.1 depicts this encapsulation. The TCP/IP Network Model defines the following layers:

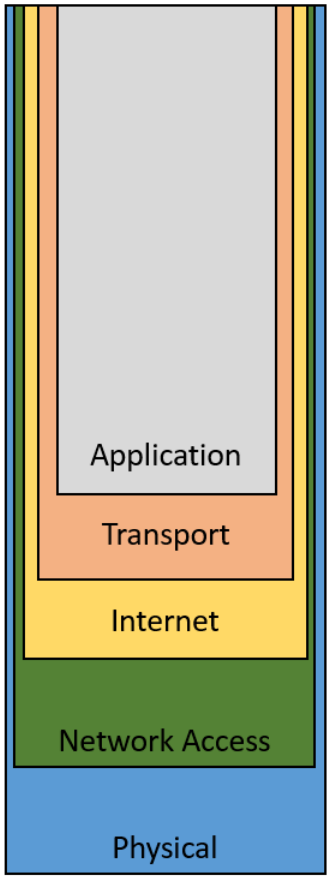


Figure 4.1: Graphical representation of encapsulation and layers of the TCP/IP Network Model.

a) Application layer. It contains the logic required to support the various applications. For each different type of application, a particular internal structure is employed.

b) Transport layer. Also defined by Stalling as host-to-host layer[72], it provides an end-to-end delivery service that might have reliability mechanisms or not.

c) Internet layer. It provides procedures to allow data traverse multiple interconnected networks, using the Internet Protocol (IP).

d) Network access layer. Also called data link layer, it provides mechanisms for accessing and routing data across a network between two devices attached to the same network.

e) Physical layer. Covers the physical interface between a data transmission device and a transmission medium or network.

There are different protocols for each layer; some layers define many protocols (application layer) and others define few protocols (internet layer). Each protocol defines how data is organized inside a layer. Most of them define two sections: 1) a header, to store control information; and 2) a body, containing application payload or encapsulation of an upper layer. Fig. 4.2 depicts this generic structure.

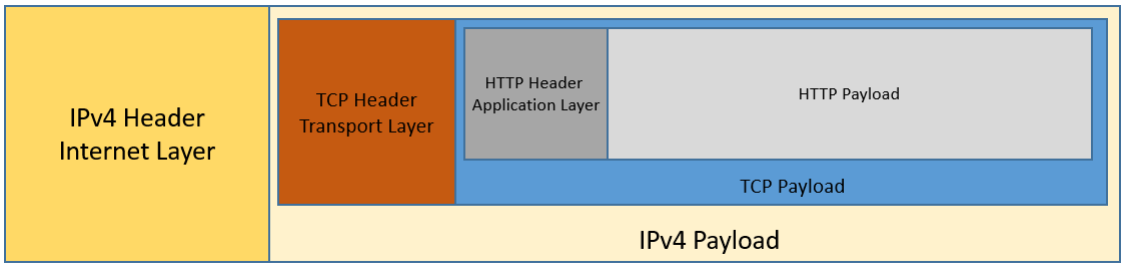


Figure 4.2: Each TCP/IP protocol defines how data is organized inside a layer. Most of them define two sections: 1) a header, to store control information; and 2) a body, containing application payload or encapsulation of an upper layer.

4.3.2 Dataset Fields

The dataset proposed has eleven fields. Table 4.1 presents each of them with the following information:

a) Field: the name of the field in the dataset proposed

b) Layer: the TCP/IP Network layer where the data belongs

c) Header field: the name of the field in the packet captured

d) View: the name of the view(s) using this field.

TABLE 4.1. FIELDS OF DATASET

Dataset Field	Layer	Header Field	Views(s)
Far-Hardware-address	Network Access	Source Address / Destination Address	Remote Host and Remote Services
Local-Hardware-address	Network Access	Source Address / Destination Address	Remote Host and Remote Services
Far-IP-address	Internet	Source Address / Destination Address	Remote Host and Remote Services
Protocol	Internet	Protocol	Remote Services
Far-Port	Transport	Source Port / Destination Port	Remote Services
Bytes	Internet	Length	Remote Host and Remote Services
HTTP-Host	Application (HTTP)	Host	Web Host HTTP URLs
HTTP-URL	Application (HTTP)	URL	HTTP URLs
HTTP-METHOD	Application (HTTP)	Method	HTTP URLs

The fields User and Timestamp are included with the purpose of allowing data analysis in a future work. Every captured packet provides all the data required to fill each of the remaining fields, taking into consideration that the HTTP-based fields might be empty if the packet captured is not HTTP. The Far and Local Hardware Address fields are extracted with the purpose of knowing the direction of the packet: incoming or outgoing. If incoming, the value for Far-IP-address field is taken from Header Field Source Address at the internet layer; otherwise, it is taken from the Destination Address, same layer. The same logic is applied for Far-Port field.

4.4. Experimental Results

A computer application that captures the network traffic was developed using Java SE 1.8. JNetPcap4 library was employed for packet capture, decoding and data extraction. In order to allow offline analysis of the information, this application stores the traffic in one CSV file for each 10 MB of data, where each line of the file denotes a captured packet and contains only the corresponding values for each of the fields introduced in Section 4.2.2.

The campus area network used in this experiment has a 16-bit network; it has a Windows domain controller and uses a HTTP Proxy. The campus applications include web-apps and remote desktop apps. The email service is provided by Microsoft Exchange Server which is hosted outside of the campus network.

The traffic on one host was captured. Since the network allows mobility and the host was a laptop, in some periods of time it was attached to the network over a wired connection and in some others over a wireless connection. At the wired network, a gigabyte Ethernet connection was used and the host has a static IP address with a 24-bit mask. At the wireless network, it has a dynamic IP address with a 22-bit mask configured by DHCP.

The traffic was captured during four labor days. The size of the traffic captured was 2.56 gigabytes, involving 4'355,262 packets. Table 4.2 presents general statistics about the capture.

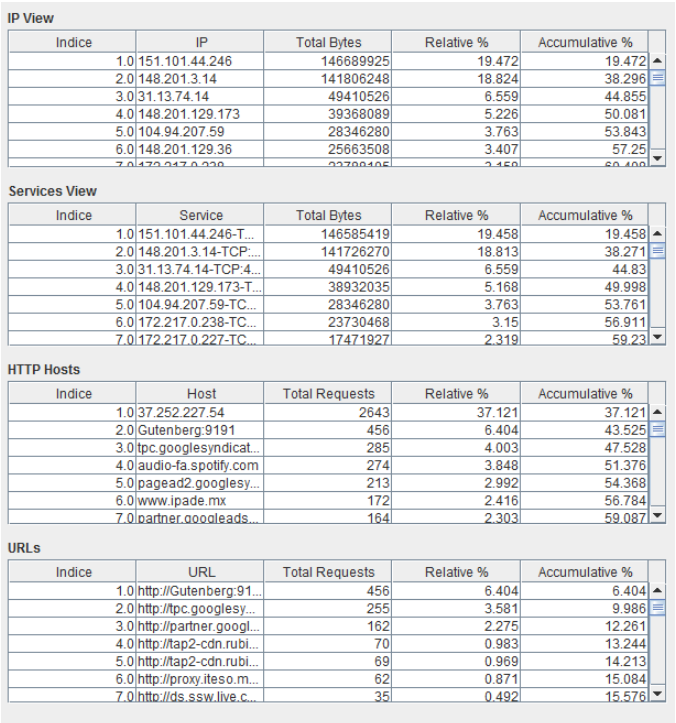
Each of the views was constructed in the form of a table. Fig. 4.3 shows examples of the tables that represent each of the views.

A timeframe of 24 hours was selected to build the user network profile. In order to select the number of top elements for each view, we first selected the number of elements that represents around 90% of data, using the amount of bytes for remote host communication and remote services accessed views, and number of request for web hosts visited and HTTP URLs accessed views. The number of elements of each view at each timeframe are exposed at Table 4.3. Based on this information 53, 60 75, 3103 were the number of top elements selected for IP view, service view, host view and URL view respective, which represents the average.

A similarity matrix was constructed to compare the user network profile of each one of the four days. Tables 4.4 to 4.7 shows the similarity matrix of each view; the number represents the percentage of equal elements between two days. In average, the similarity between profiles is 26.59%.

TABLE 4.2. GENERAL STATISTICS OF CAPTURE TRAFFIC

	Day 1		Day 2		Day 3		Day 4		Total
Total Bytes (MB)	885	34%	718	27%	494	19%	532	20%	2,629
Count Packets	1,380,498	32%	1,248,241	29%	904,538	21%	821,985	19%	4,355,262
Different IPs	1,111	49%	1,033	46%	841	37%	658	29%	2,260
Different Services	8,078	25%	9,548	29%	10,806	33%	6,820	21%	32,641
Different HTTP Hosts	382	51%	338	45%	216	29%	209	28%	743
Different URLs	5,980	43%	5,115	37%	1,395	10%	1,778	13%	13,768



IP View

Indice	IP	Total Bytes	Relative %	Accumulative %
1.0	151.101.44.246	146689925	19.472	19.472
2.0	148.201.3.14	141806248	18.824	38.296
3.0	31.13.74.14	49410526	6.559	44.855
4.0	148.201.129.173	39368089	5.226	50.081
5.0	104.94.207.59	28346280	3.763	53.843
6.0	148.201.129.36	25663508	3.407	57.25

Services View

Indice	Service	Total Bytes	Relative %	Accumulative %
1.0	151.101.44.246-T...	146585419	19.458	19.458
2.0	148.201.3.14-TCP:...	141726270	18.813	38.271
3.0	31.13.74.14-TCP:4...	49410526	6.559	44.83
4.0	148.201.129.173-T...	38932035	5.168	49.998
5.0	104.94.207.59-TC...	28346280	3.763	53.761
6.0	172.217.0.238-TC...	23730468	3.15	56.911
7.0	172.217.0.227-TC...	17471927	2.319	59.23

HTTP Hosts

Indice	Host	Total Requests	Relative %	Accumulative %
1.0	37.252.227.54	2643	37.121	37.121
2.0	Gutenberg:9191	456	6.404	43.525
3.0	tpc.googlesyndicat...	285	4.003	47.528
4.0	audio-fa.spotify.com	274	3.848	51.376
5.0	pagead2.googlesy...	213	2.992	54.368
6.0	www.ipade.mx	172	2.416	56.784
7.0	partner.googleads...	164	2.303	59.087

URLs

Indice	URL	Total Requests	Relative %	Accumulative %
1.0	http://Gutenberg:91...	456	6.404	6.404
2.0	http://tpc.googlesy...	255	3.581	9.986
3.0	http://partner.googl...	162	2.275	12.261
4.0	http://tap2-cdn.rubi...	70	0.983	13.244
5.0	http://tap2-cdn.rubi...	69	0.969	14.213
6.0	http://proxy.iteso.m...	62	0.871	15.084
7.0	http://ds.ssw.live.c...	35	0.492	15.576

Figure 4.3: Example of the tables that represents each of the views.

TABLE 4.3. NUMBER OF ELMENTS THAT REPRESENTS 90% OF DATA

	Day 1	Day 2	Day 3	Day 4	Avg.
IP View	56	65	56	33	53
Service View	66	73	64	37	60
Host View	89	64	76	69	75
URL View	5241	4403	1215	1553	3103

4.5. Conclusions

In this chapter, we have presented a 4-view network user profile built from real-time network traffic captured at the end-user host.

The main differences from other approaches that generate user profiles based on network traffic are the following: 1) the network traffic is captured at the end-user host or at a near network point, and 2) the use of four different views to build the user profile: IP Address, Service reached, HTTP Host and URL.

We captured traffic at one host during four labor days and constructed a network profile of the same user for each day; then we compared the network profiles with each other to find similarities. We observed that the IP, Services and Hosts views showed all a similarity around 30%; however, the URL view showed less than 4% of similarity.

TABLE 4.4. IP VIEW - SIMILARITY MATRIX(%)

. AVERAGE 37.20%

	Day 1	Day 2	Day 3	Day 4
Day 1	-	44.64	41.07	30.36
Day 2	44.64	-	44.64	26.79
Day 3	41.07	44.64	-	35.71
Day 4	30.36	26.79	35.71	-

TABLE 4.5. SERVICE VIEW - SIMILARITY MATRIX(%)

. AVERAGE 34.72%

	Day 1	Day 2	Day 3	Day 4
Day 1	-	43.33	38.33	30.00
Day 2	43.33	-	41.67	25.00
Day 3	38.33	41.67	-	30.00
Day 4	30.00	25.00	30.00	-

TABLE 4.6. HTTP HOST VIEW - SIMILARITY MATRIX(%)

. AVERAGE 30.44%

	Day 1	Day 2	Day 3	Day 4
Day 1	-	33.33	33.33	26.67
Day 2	33.33	-	30.67	21.33
Day 3	33.33	30.67	-	37.33
Day 4	26.67	21.33	37.33	-

TABLE 4.7. HTTP URL VIEW - SIMILARITY MATRIX(%)

. AVERAGE 3.98%

	Day 1	Day 2	Day 3	Day 4
Day 1	-	2.96	2.87	3.38
Day 2	2.96	-	1.48	2.26
Day 3	6.38	3.30	-	6.88
Day 4	5.91	3.94	5.40	-

5. Profiling Network Traffic for Internal Security Using Rankings Similarity Measures

The 4-view profile, presented in Chapter 4, is a network profile built with traffic captured at the host or a near point from it. This will help to determine if a host is having or not the expected behavior, i.e., if the host is under attack or not.

In order to compare two profiles, we propose to treat each view of the 4-view profile as a TopK ranking, and to use a similarity measure for TopK rankings in order to compare them.

In this work, we propose a 3-phase methodology to calculate a similarity between real-time captured traffic –current behavior– and traffic captured a priori within a period of normal behavior –expected behavior– (see 5.2).

In order to test the accuracy of this methodology, we build the normal-behavior profile of a user, and then calculate the similarity factor between this same user and two others.

5.1. Related work on Users Profile and Network Security

The usage of network user profiles for representing the network behavior has been part of the research about computer network security.

Kihl et al.[43] present a work about traffic analysis and characterization of Internet users to help understanding the Internet usage and the demands on broadband access. They use a commercial tool for capturing and classifying traffic according to the Internet protocols and applications. This work concludes that the usage of Internet has changed from traditional WWW requests to a more complex use. Their results about Internet usage in 2010 indicate is that most of this traffic comes from: sharing files protocols (74%), media streaming (7.6%), and web-traffic (5.5%). The traffic for this work was collected from a Swedish municipal FTTH network.

Sing et al.[70] present an intrusion detection technique using network-traffic profiling and an online sequential extreme machine-learning algorithm. The proposed methodology uses a profiling procedure, called alpha profiling, that creates profiles on the basis of protocol and service features; and a second profiling process, beta profiling, where the alpha profiles are grouped to reduce the

number of profiles. The authors made three different experiments: 1) using all features and alpha profiling, 2) using only some features and alpha profiling, and 3) using only some features, alpha profiling and beta profiling. The best results were obtained from the last experiment using both profiling methods. The dataset used for this work was NSL-KDD.

A work that builds profiles of network prefixes instead of users is presented by Qin et al[64]. They propose aggregating traffic based on network prefixes in order to reduce the amount of data to be processed, and then calculate clusters using a k-means algorithm. Qin found that similar users produce similar traffic; with this information, decisions about security and management can be taken. The traffic used for this work was captured at CERNET backbone.

A similar work is presented by Xu et al[82], who proposed a methodology that analyzes Internet traffic. This methodology first constructs bipartite graphs; after this, it generates one-mode projections; then, it builds a similarity matrix and generates cluster with a spectral clustering method; finally, it analyzes the clusters. The traffic used in this work was captured at the backbone of a large Internet service provider, aggregating the information using 24-bit length prefix networks, and the network 5-tuple.

As we can see, all the works discussed above have used the traffic captured at a far point from the end-user host, even outside the local network of the user, leaving unattended the internal network security. In addition, the usage of profiles has proved viable to either identify or specify network behaviors.

5.2. Methodology Proposed

The proposed methodology has the goal of determining how similar the traffic captured in real time is to the traffic captured a priori in a controlled environment, i.e., the normal-behavior. This methodology uses the 4-view network profile profile for grouping the captured traffic.

5.2.1 Build User Normal-Behavior

This phase builds a user network profile called normal-behavior, which will be used as reference for calculating the similarity factor. Fig. 5.1 shows the overall process at this phase.

Network traffic, $P_{SE,x}$, is captured from user x during a period of time T in a secure environment

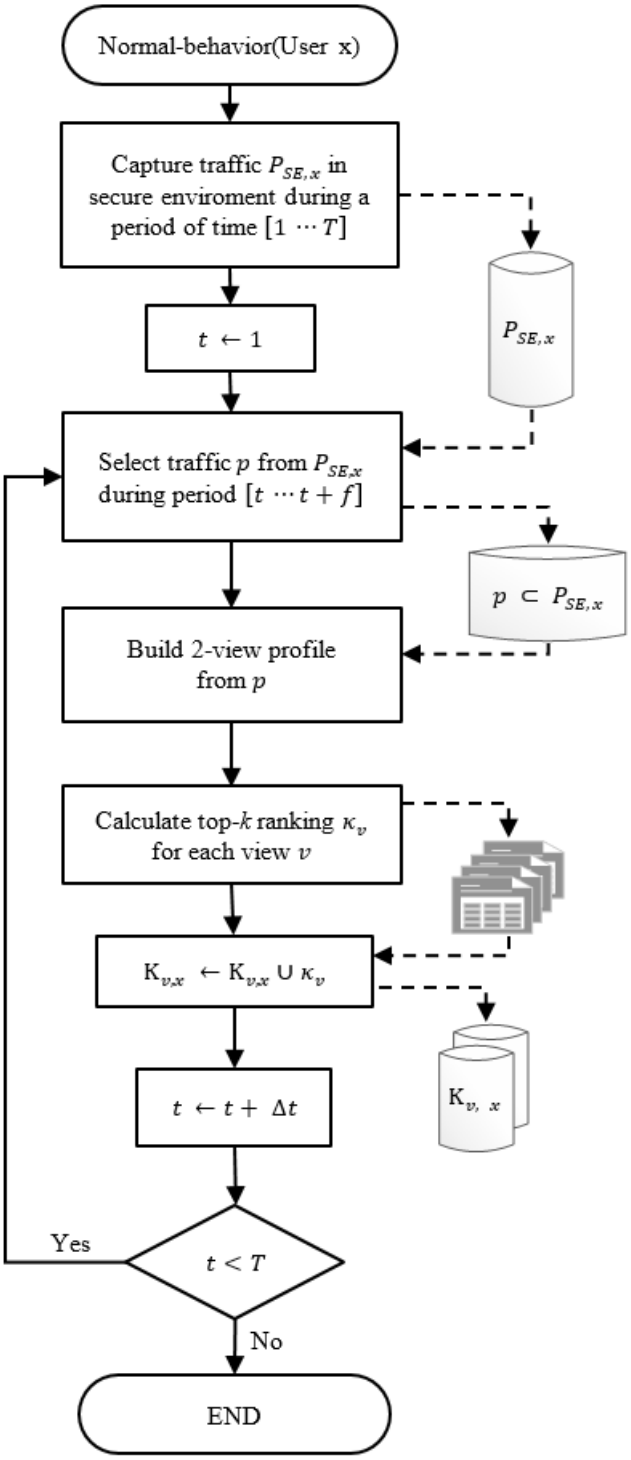


Figure 5.1: Building the normal-behavior profile of user x during a period of time T.

at which we can guarantee that the host will be used only by the expected user and there is no malware, virus, trojan or any other malicious software installed. The period of time T must be long enough to make sure that all the habitual tasks of the user are registered.

From $P_{SE,x}$ is selected a subset p that corresponds to the period of time $[t \cdots t + f]$. With traffic p, the 4-view profile is built, and from each view v, a TopK ranking is calculated and stored in $K_{v,x}$. This process is repeated while t is less than T. At the end of each iteration, t increments in Δt, a value smaller than f in order to produce overlaps in timeframes.

An extraction of a TopK ranking from the IP view is illustrated in Fig. 5.2, where $k = 10$ and five time frames are listed.

Timeframe	Top 1	Top 2	$\cdots$	Top 10
$[t \cdots t + f]$	148.201.129.173	148.201.140.219	$\cdots$	148.201.140.50
$[t + \Delta t \cdots t + \Delta t + f]$	148.201.129.173	148.201.140.219	$\cdots$	148.201.140.50
$[t + 2\Delta t \cdots t + 2\Delta t + f]$	148.201.129.173	148.201.140.148	$\cdots$	148.201.140.98
$[t + 3\Delta t \cdots t + 3\Delta t + f]$	132.245.44.22	148.201.140.148	$\cdots$	148.201.129.43
$[t + 4\Delta t \cdots t + 4\Delta t + f]$	148.201.140.148	148.201.140.219	$\cdots$	148.201.129.43

Figure 5.2: Example of a TopK ranking from IP View.

5.2.2 Capture Regular-Traffic

In this phase, regular-traffic, $P_{rt,x}$ is captured in real time from the user host during a period of time $[t \cdots t + f]$, and for each Δt increment. Using this traffic, the 4-view profile is built and, for each view v, a TopK ranking is calculated. Using each of these TopK rankings, the best similarity factor against every TopK ranking in $K_{v,x}$ is found, as it is described in the next section. Fig. 5.3 shows the flow diagram of this process.

5.2.3 Calculate Best Similarity Factor

The purpose of this phase is to find out if the real-time traffic captured during a period of time $[t \cdots t + f]$ resembles to any traffic within the records of the normal behavior captured during a period of time of length f. Thus, a similarity factor is calculated between the TopK ranking of the

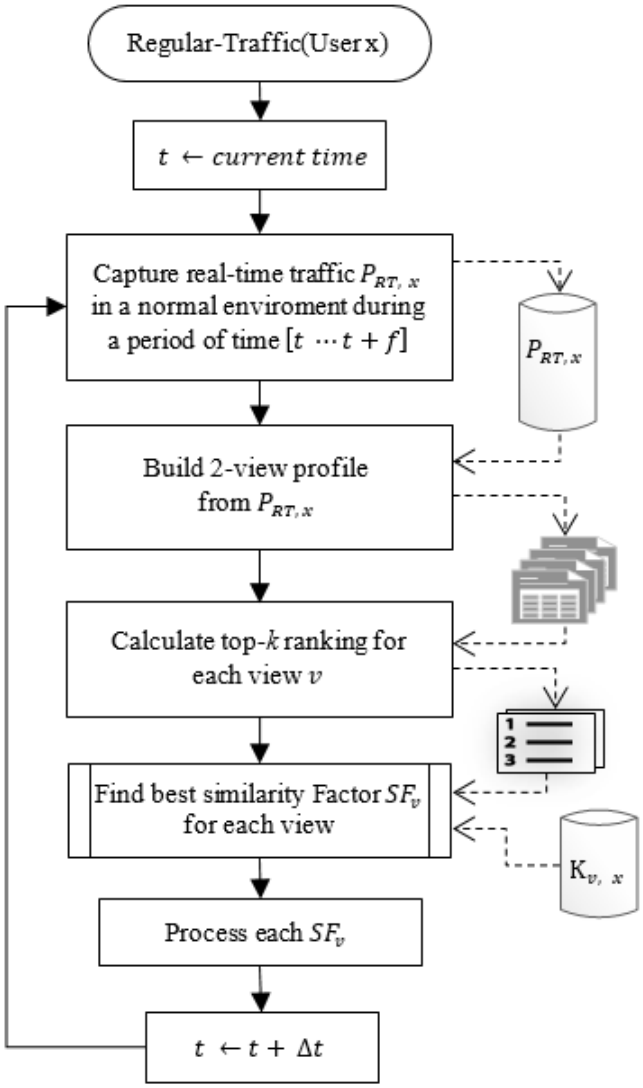


Figure 5.3: Process of capture real-time traffic from user x.

real-time traffic $(\kappa_{v,x'})$ and each of the TopK rankings stored in $K_{v,x}$. The best similarity factor is found for each view. According to this value, a decision might be taken about the correspondence of the current network traffic with the expected one. Fig. 5.4 shows a diagram of this process.

To compare each pair of TopK, it is necessary to use a ranking similarity measure, but with the peculiarity that these TopK rankings are non-conjoint rankings, this means that not all elements of one of them are present in the other. Thus, we have explored two different measures:

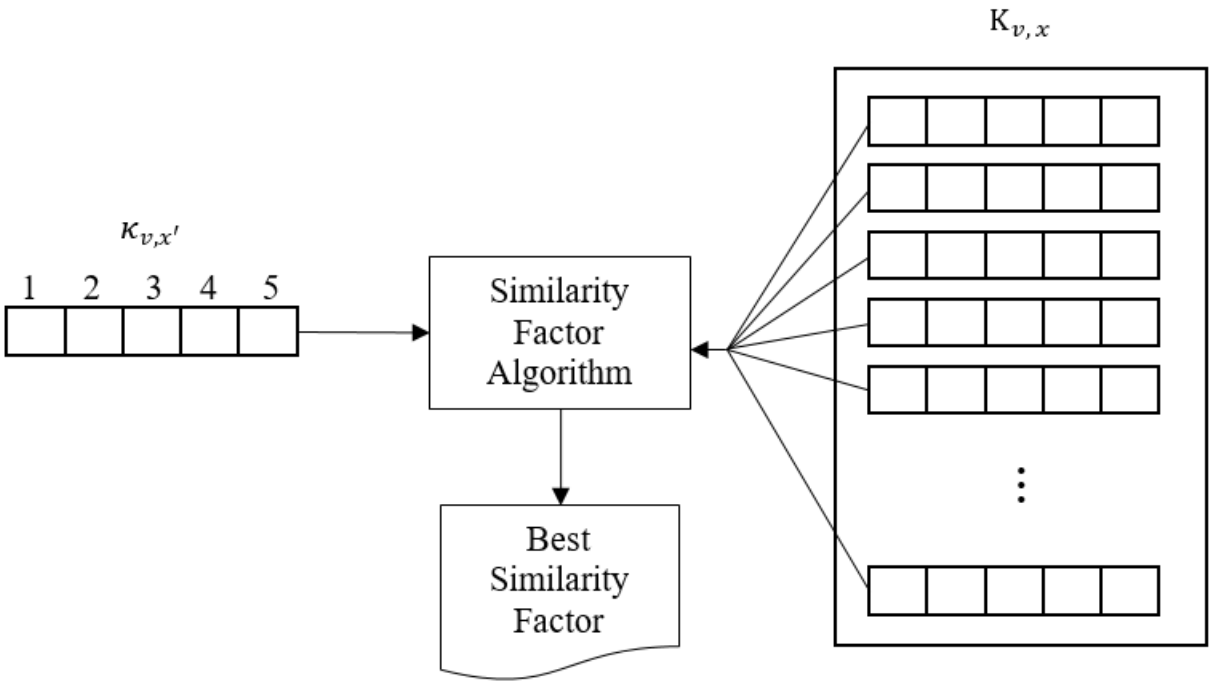


Figure 5.4: Calculating best similarity factor

Spearman's rho

Spearman's rho[28] is the distance between two permutations, formally:

$$\rho(\sigma_1, \sigma_2) = \left(\sum_{i=1}^{n} | \sigma_1(i) - \sigma_2(i) |^2 \right)^{1/2} \tag{5.1}$$

In the case of non-conjoint rankings, we use the Fagin approach that considers that both TopK rankings are a bijection from a domain D, that contains all the elements. It can be assumed that every non-present element in TopK list is ranked in some position after k^{th}[28].

Based on this, we define that the distance for a non-present element is k. The new formula is:

$$\rho(X, Y) = \left(\sum_{i=1}^{k} | \sigma_X(i) - \sigma_Y(i) |^2 \right)^{1/2} \tag{5.2}$$

where X and Y are TopK rankings from the same domain, $\sigma_K(i)$ is the rank of element i in a TopK ranking; in case element i is not present at this list, the value of $\sigma_K(i)$ is K, where K is the number of elements of both lists.

Finally, in order to have similar results to the second measure, we re-write the formula to get a normalized value between 0 and 1, where 0 denotes identical TopK rankings, and 1 denotes totally different TopK rankings. Formally:

$$\rho(X,Y) = \frac{1}{k^3}\left(\sum_{i=1}^{k} \mid \sigma_X(i) - \sigma_Y(i) \mid^2\right) \tag{5.3}$$

Average Overlap

The second measure is based on the work by Webber[79] called average overlap. This measure calculate for each $d \in 1 \cdots k$, the overlap at d, and then averages those overlaps to derive the similarity measure. Formally, the average overlap between two TopK rankings can be expressed as:

$$AO(S,T,k) = \frac{1}{k}\sum_{d=1}^{k} A_{S,T,d} \tag{5.4}$$

where S and T are top lists of k number of elements, and $A_{S,T,d}$ is defined as:

$$A_{S,T,d} = \frac{\mid S_{:d} \cap T_{:d} \mid}{d} \tag{5.5}$$

where $S_{:d}$ and $T_{:d}$ represents the each one of the rankings with only d top elements. Fig. 5.5 shows an example where the AO of two top-5 lists is calculated.

Finally, in order to have similar results to the first measure, we calculated the complement

d	$S_{:d}$	T: d	$A_{S,T,d}$	$AO(S,T,d)$
1	$< a >$	$< x >$	0.0000	0.0000
2	$< ab >$	$< xc >$	0.0000	0.0000
3	$< abc >$	$< xcb >$	0.6667	0.2222
4	$< abcd >$	$< xcby >$	0.5000	0.2917
5	$< abcde >$	$< xcbye >$	0.6000	0.3534

Figure 5.5: Similarity calculation of two top-5 lists using average overlap measure.

where 0 denotes identical TopK rankings, and 1 denotes totally different TopK rankings. Formally:

$$AO(S,T,k) = 1 - \frac{1}{k}\sum_{d=1}^{k} A_{S,T,d} \tag{5.6}$$

5.3. Experiments and Results

5.3.1 Experiment Setup

The experiment was done at a campus area network that has a 16-bit network; it has a Windows domain controller and uses an HTTP proxy. The campus applications include web-apps and remote desktop apps. The email service is provided by Microsoft Exchange Server, which is hosted outside of the campus network.

The target users are full-time professors, who have a computer with two types of access to the network: a wired access with a static IP address and a wireless access with a dynamic IP address.

For this experiment, one full-time professor was selected as User A, for whom we have generated his normal-behavior traffic $_A$ and then captured real-time traffic κ_A. With the goal of validating the methodology proposed, two other professors from the same academic department were selected as Users B and C, and generated real-time traffic κ_B and κ_C.

Different values of the parameters were evaluated: $f = 1, 5, 10min$, $\Delta t = 10, 30, 60sec$, the number of elements selected for the TopK rankings, $k = 10, 25, 50, 100$, and both measure functions.

The best combination $< t, \Delta t, k, Function >$ was: $t = 5min$, $\Delta t = 10sec$, $k = 10$, $Average\ Overlap$. It maximizes the differences between $AO(_A, \kappa'_A)$ and both $AO(_A, \kappa_B)$ and $AO(_A, \kappa_C)$.

5.3.2 Capturing and Processing Traffic

During 4 labor-days, traffic was captured to characterize normal-behavior from User A's computer. Before start capturing we made a check-up to validate that the computer has not installed any malicious software. During all this period only this user had access to the computer. The size of the traffic captured was 2.56 gigabytes, involving more than four millions of packets. All the packets were processed as describe in Section III.A. On different days, 8-hour real-time traffic was captured from the computers of users A, B and C ($\kappa_{A'}$, κ_B,κ_C). This traffic was processed as

described in Section 5.2.2.

5.3.3 Similarity Calculation

For each TopK, $\kappa_{A'}$, κ_B and κ_C the best similarity factor with respect to A was found and plotted; see Fig. 5.6 (IP view) and 5.7 (service view). We can see that User A exhibited a higher similarity to his own normal behavior than user B and C, as expected. Fig. 5.8 (IP view) shows the similarity factors as a histogram.

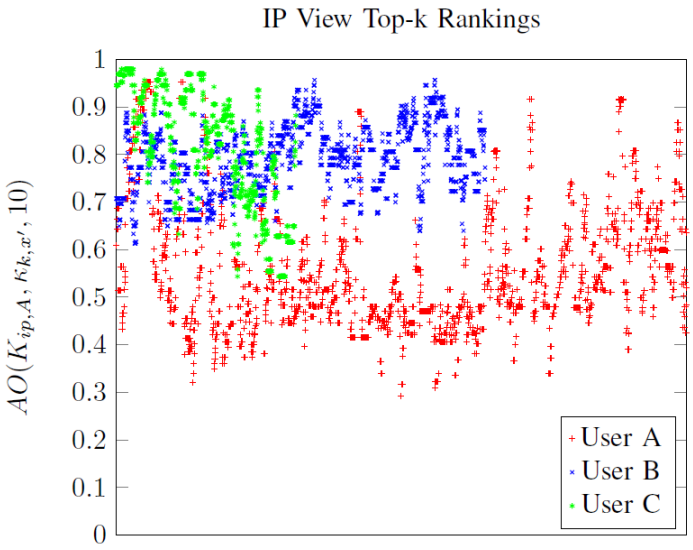


Figure 5.6: Similarity Factor of IP View between $\kappa_{A'}$, κ_B and κ_C against K_A

Since TopK rankings are independent from each other, we can order the similarity factors by value to get a clearer graph. Fig. 5.9 (IP view) and 5.10 (service view) show all the similarity factors order by value.

HTTP Host view presented poor similarity factors for all the users, so this view could not allow us to differentiate user A from users B and C (see Fig. 5.11); similar results occur with HTTP URL view.

5.4. Conclusions

In this chapter, we have proposed a methodology to determine how similar is the traffic captured in real-time at a user host $\kappa_{v,x'}$, to a previously captured traffic that we called normal-behavior

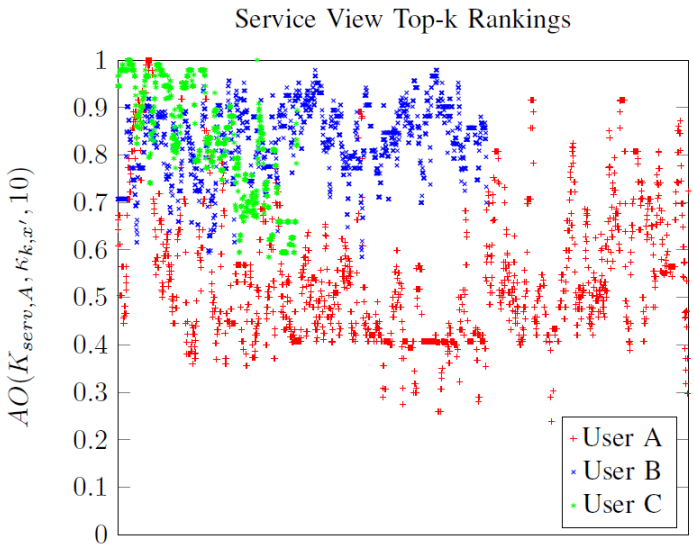


Figure 5.7: Similarity Factor of Service View between $\kappa_{A'}$, κ_B and κ_C against K_A

$K_{v,x}$, at the same host or another. Also, we presented the results of the implementation of such methodology. The similarity was calculated for each view from the 4-view profile.

A preliminary early conclusion from this work is that the IP and Service views allow to determine if the captured traffic corresponds to the expected user. In the presented graphs we can see how the User A has better similarity factors than users B and C. The HTTP Host and HTTP URL views did not present the same behavior than IP and Services view, in many time-frames the similarity factor was better for User B and C than for User A. This might be due to the low volume of HTTP traffic, most of the web traffic is with HTTPS protocol that does not allow the capture process read the request host and URL.

From the results we can see that non TopK ranking from $\kappa_{v,x}$ is identical to any TopK from $K_{v,x}$, i.e. the similarity factor is 0.0. A possible reason for this is that multiple IP addresses can be configured by the same host or service, or the user really does not do exactly the same all the time. But we consider that the similarity factors obtained are good enough to differentiate between the expected user from the others.

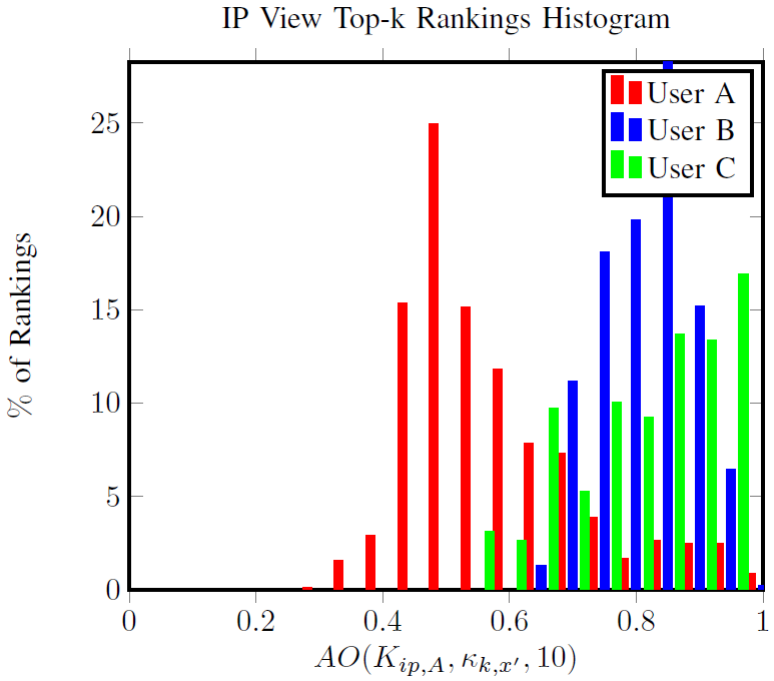


Figure 5.8: Histogram of Similarity Factors from IP View between $\kappa_{A'}$, κ_B and κ_C against K_A

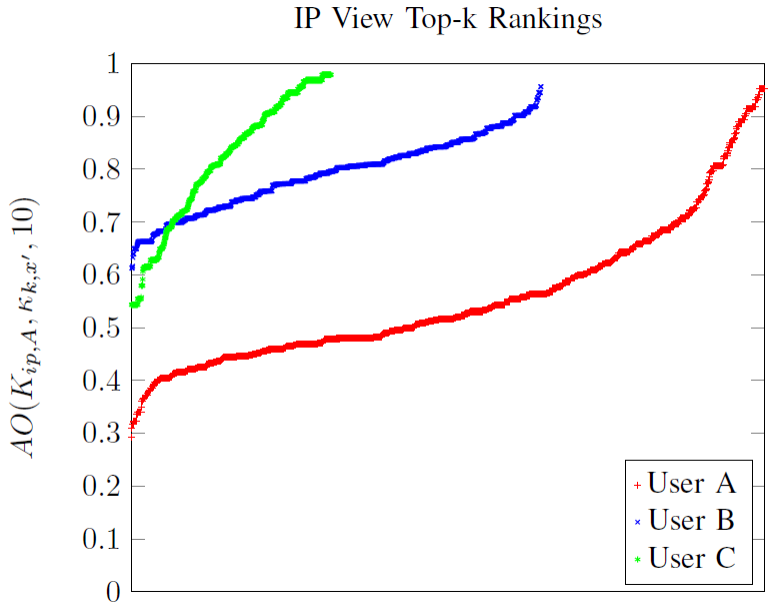


Figure 5.9: Similarity Factor of IP View between $\kappa_{A'}$, κ_B and κ_C against K_A ordered by value.

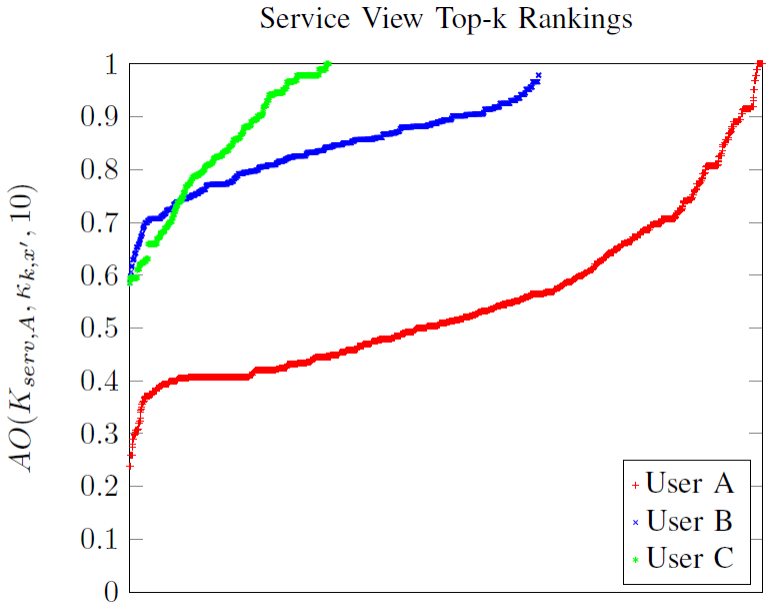


Figure 5.10: Similarity Factor of Service View between $\kappa_{A'}$, κ_B and κ_C against K_A ordered by value.

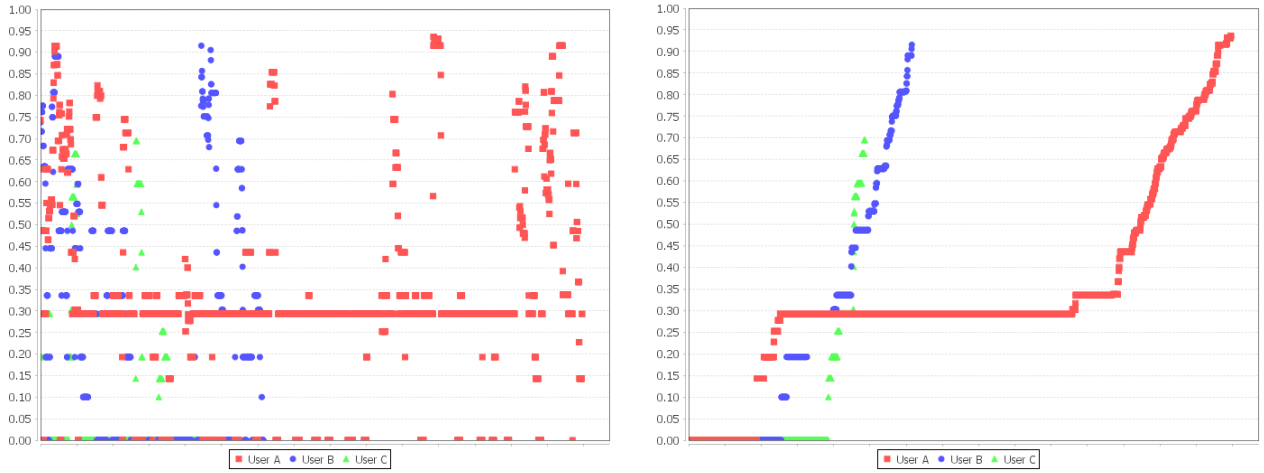


Figure 5.11: Similarity Factor of HTTP Host View between $\kappa_{A'}$, κ_B and κ_C against K_A ordered by: a) capture timestamp; b) by value.

6. Comparing User Network Profiles by Means of TopK Ranking Similarity Measures

Network-user profiling has been used to detect unknown attacks by trying to identify an unexpected network behavior, which is a common symptom of security problem. [64, 70, 82]

The network-user profiling based on TopK ranking presented in Chapter 5 allows us to measure how similar a traffic sample is to a previously captured traffic that characterizes a specific user. In order to generate user profiles, this profiling technique calculates the TopK ranking of accessed services, using the total amount of bytes transferred to build the ranking. It uses the average overlap measure[79] to compare traffic against the profiles.

In this chapter, we have built the user profile of five full-time professors in a campus area network by capturing network traffic during a full labor week. Then, for each of the professors, we captured traffic again during two different periods of time and compared them with each of the user network profiles. In a second experiment, we intend to predict the owner of a small amount of traffic that is captured in real time.

6.1. TopK network user profile

The network user profiling technique used is based on the one presented in Chapter 5, which treats the network traffic as a ranking. From this point, we decided to use only the accessed services by the user; this means that the ranking list is built using the 3-tuple <remote IP address, protocol, remote port>. We use the average overlap[79] method to compare the profiles. Since we compare only the TopK elements of each ranking, the user profile contains only the TopK rankings, and it is named TopK Network user profile (TkNP).

6.1.1 Building a TopK Network User Profile

From the captured network traffic of a user during a period of time T, a subset of traffic that corresponds to the period of time $[t \cdots + f]$ is selected, the accessed services are aggregated

according to the bytes transferred, and the TopK ranking is calculated and stored in K. This process is repeated while t is less than T. At the end of each iteration, t increments in Δt, a value smaller than f in order to produce overlaps in timeframes. All the TopK rankings are stored in K.

6.1.2 Comparing User Profiles

Given two TopK network user profiles P_S and P_T, the similarity of P_S with respect to P_T is defined as a list where the i^{th} element contains the highest similarity factor found between the i^{th} TopK in P_S and every TopK in P_T. This factor is calculated using the average overlap.[79]

6.2. Experminet and Results

6.2.1 Experiment Setup

The experiment was carried out in a campus area network that has a 16-bit network; it has a Windows domain controller and uses a HTTP proxy. The campus applications include web-apps and remote desktop apps. The email service is provided by Microsoft Exchange Server which is hosted outside of the campus network.

The target users were full-time professors working on a laptop with two types of network accesses: 1) a wired access with a static IP address and 2) a wireless access with a dynamic IP address; the users use their computers inside the campus regularly, but sometimes outside. Building TkNP and Capturing Datasets.

We have captured the traffic of each professor (A, B, C, D and E) during a full labor week and built their TkNPs: P_A, P_B, P_C, P_D and P_E. In a later period of time, we captured traffic from the same users and built two small TkNPs for each one: κ_{A_1} to κ_{E_1} and κ_{A_2} to κ_{E_2}. Each small TkNP contains 1000 TopK rankings randomly selected, such that no TopK ranking belongs to more than one small TkNP.

6.2.2 Comparing TkNPs and Results

We calculated the similarity between each small TkNP (κ_{A_1} to κ_{E_1} and κ_{A_2} to κ_{E_2}) and every TkNP (P_A to P_E). Fig 6.1 to Fig 6.5 plot in descending order the values of each similarity. We can

observe that the highest line on every figure corresponds to the similarity between two profiles of the same user, i.e., κ_{A_1} and P_A, κ_{A_2} and P_A, $\cdots$ κ_{E_2} and P_E. Also, both small TkNP of the same users produce similar graphs.

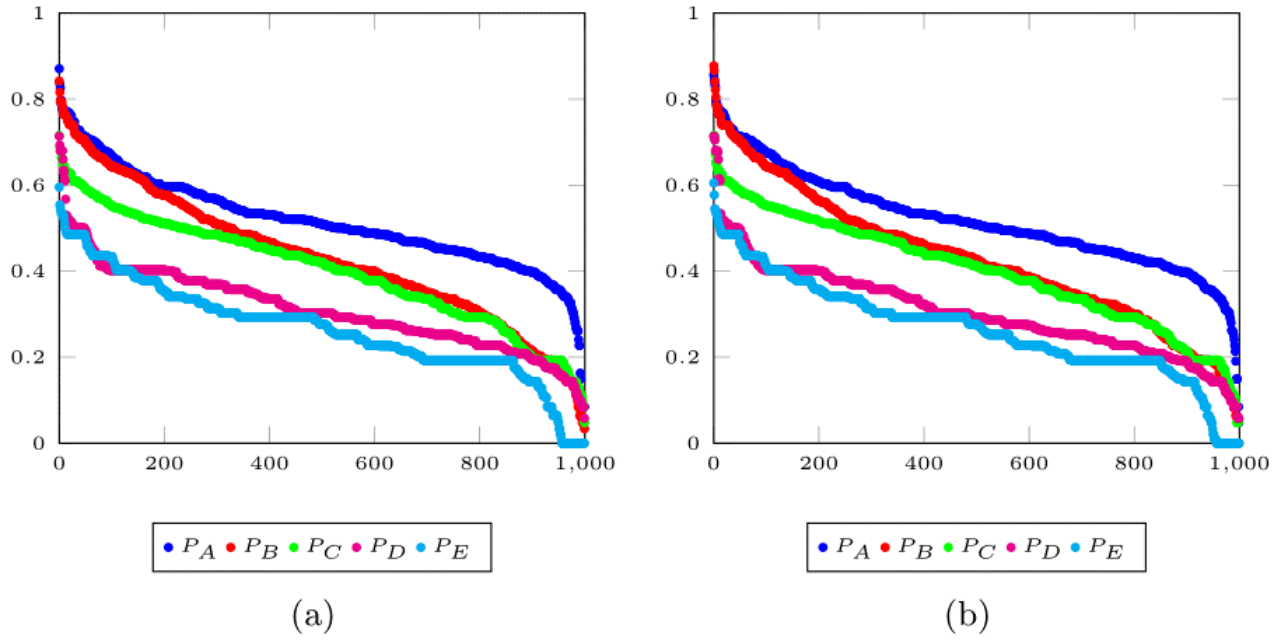


Figure 6.1: Similarities between: a) κ_{A_1} and P_A to P_E b) κ_{A_2} and P_A to P_E

Table 6.1 shows the averages of the similarity between each small TkNP and all TkNPs. We can observe that the highest average of each small TkNP is found at the column that corresponds to the same user (main diagonal). This information is also presented as radar chart in Fig 6.6.

6.2.3 Identifying Traffic Owner

The purpose of this experiment is to predict the owner of a small amount of traffic (five minutes of capture) using the TkNPs. In this context the owner will be the user (A to E) with the highest similarity to the TopK ranking built from such traffic.

We calculated the similarity between each TopK in κ_{A_1} to κ_{E_1}, as unitary TkNPs, and every TkNP (P_A to P_E). Then we labeled each TopK as the user (A to E) that corresponds to the TkNP with the maximum similarity. Table 6.2 shows the percentage of TopK ranking in each small TkNP labeled as each of the users. We can observe that the highest percentage is found at the column that corresponds to the same user (main diagonal).

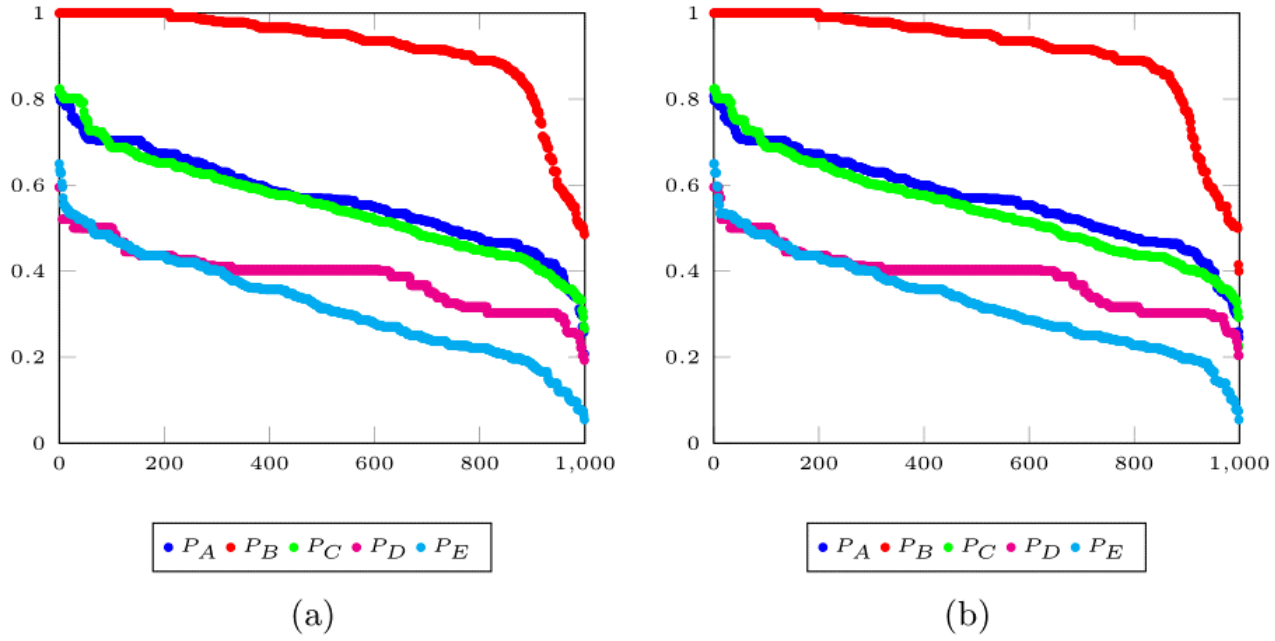


Figure 6.2: Similarities between: a) κ_{B_1} and P_A to P_E b) κ_{B_2} and P_A to P_E

6.3. Conclusions

In this work, we have compared the network traffic of five different users by building TopK network profiles and similarity measures.

Early results suggest that the method proposed to build network user profiles is viable to distinguish one user from another. In addition, we could observe that five minutes of network traffic capture is often enough to determine who the traffic belongs to.

All the experiments were carried out with two different datasets of traffic from each user, and we got similar results in both datasets. Thus, we can conclude that the methodology proposed is consistent.

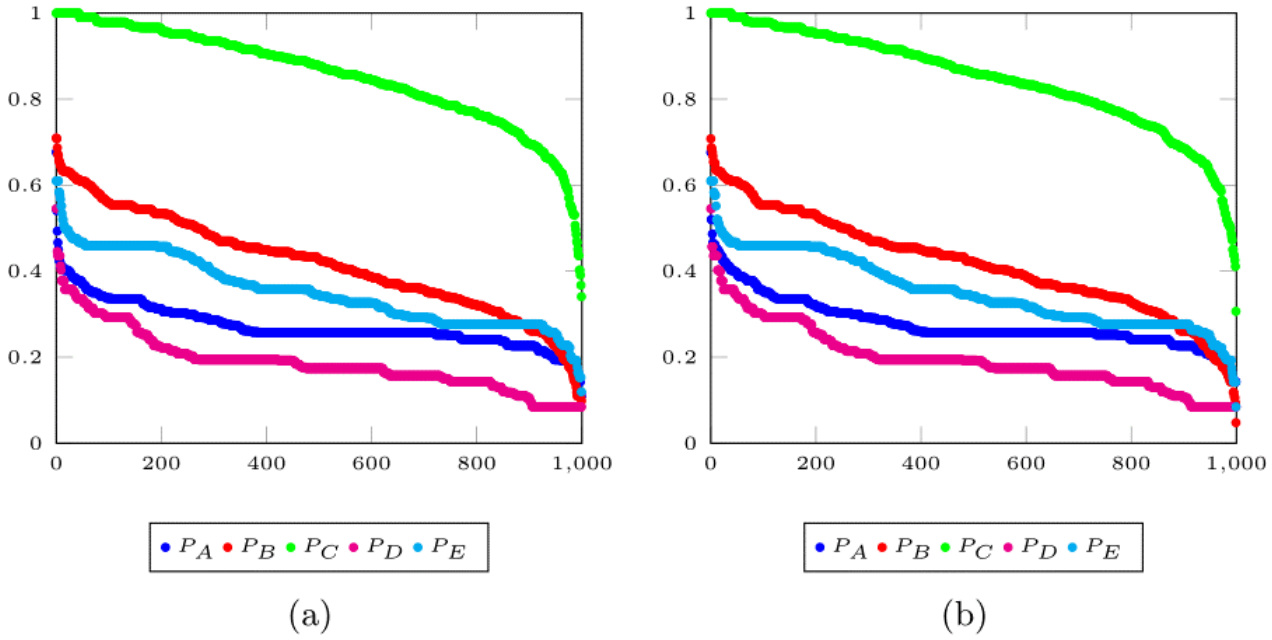


Figure 6.3: Similarities between: a) κ_{C_1} and P_A to P_E b) κ_{C_2} and P_A to P_E

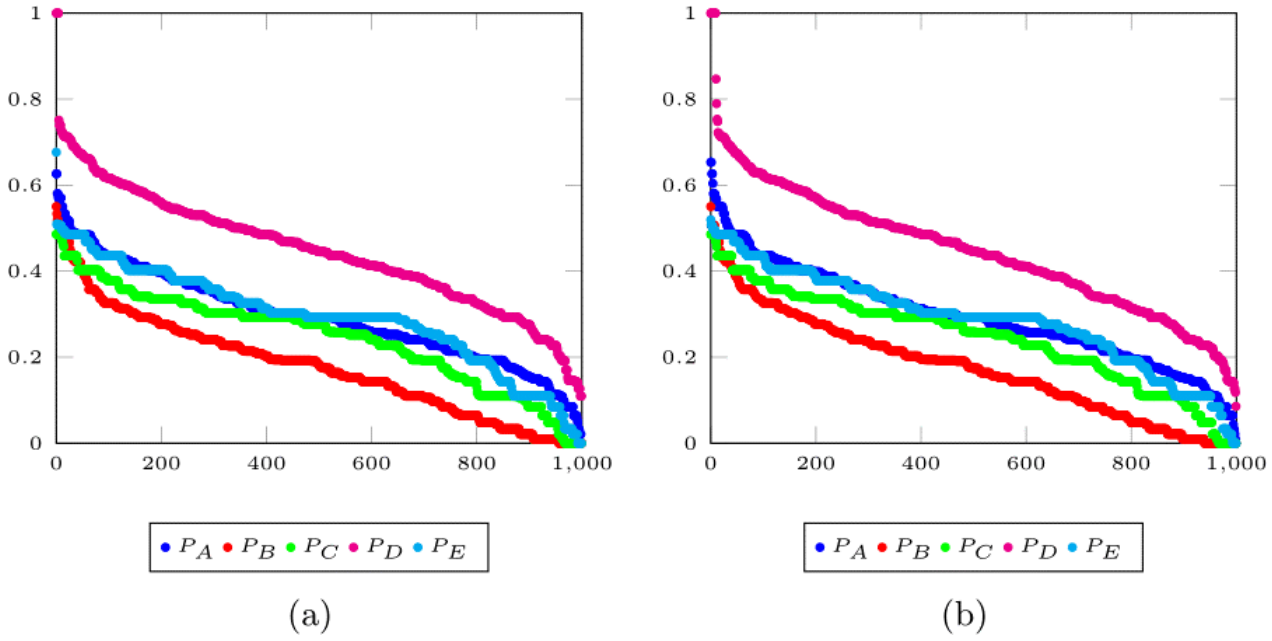


Figure 6.4: Similarities between: a) κ_{D_1} and P_A to P_E b) κ_{D_2} and P_A to P_E

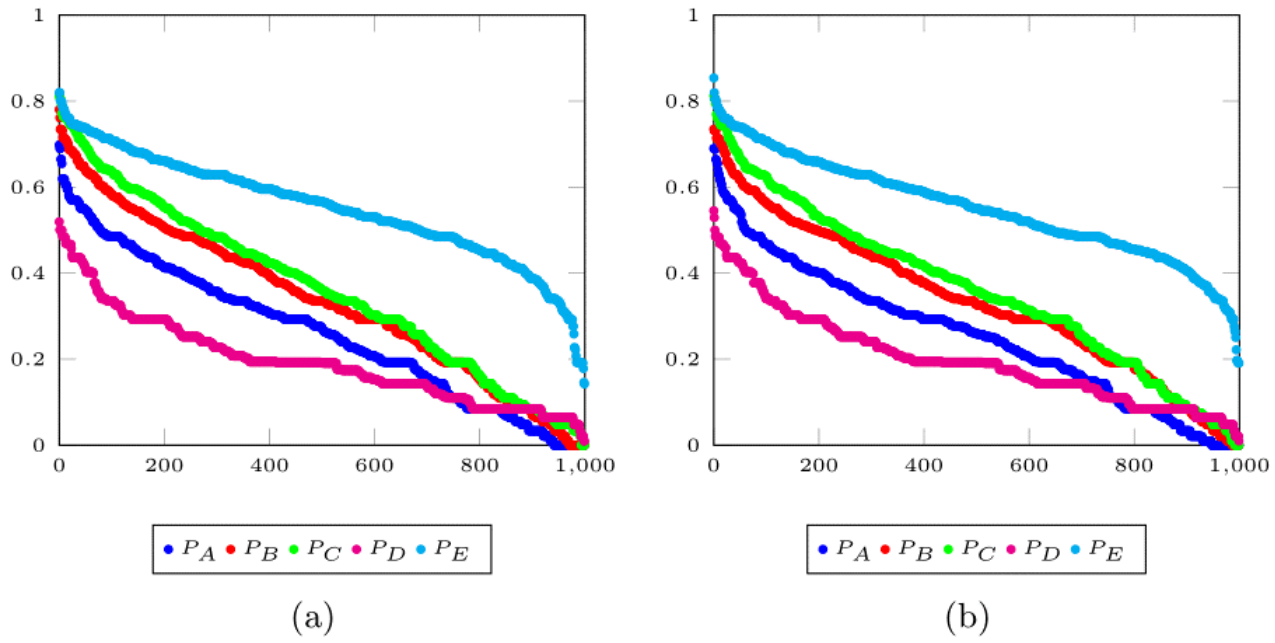


Figure 6.5: Similarities between: a) κ_{E_1} and P_A to P_E b) κ_{E_2} and P_A to P_E

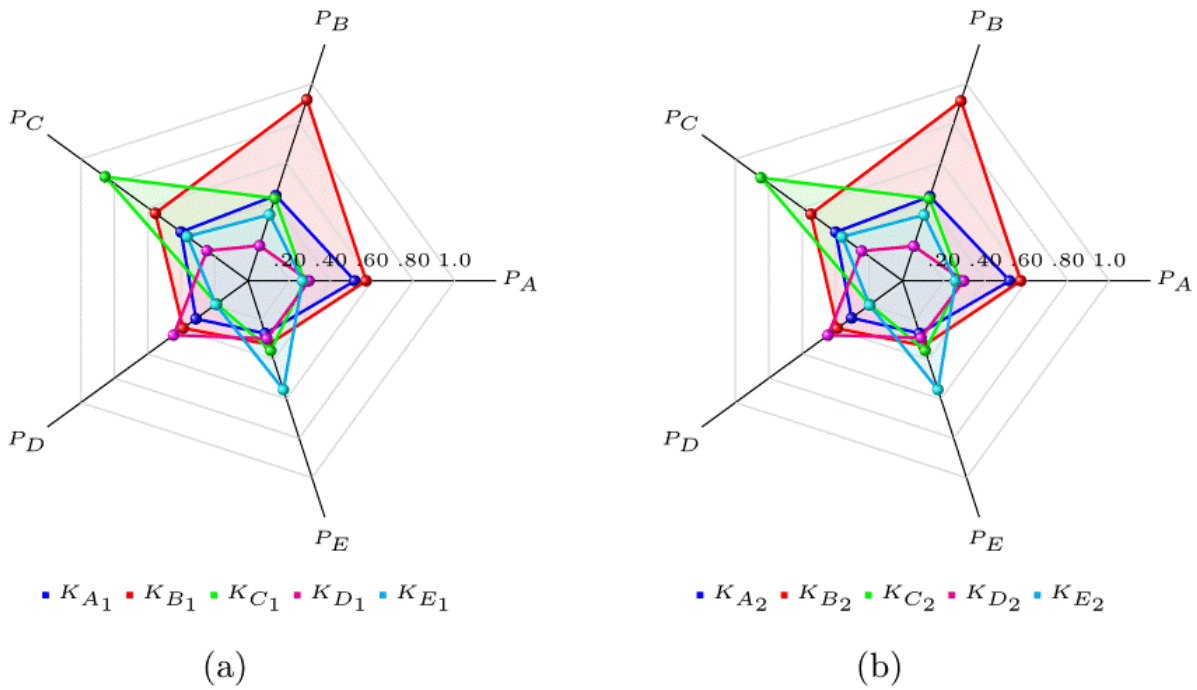


Figure 6.6: Average Similarities between: a) κ_{A_1} to κ_{E_1} and P_A to P_E b) κ_{A_2} to κ_{E_2} and P_A to P_E

TABLE 6.1. AVERAGE SIMILARITIES

	P_A	P_B	P_C	P_D	P_E
κ_{A_1}	0.516	0.434	0.401	0.312	0.267
κ_{A_2}	0.518	0.429	0.400	0.305	0.266
κ_{B_1}	0.571	0.920	0.555	0.389	0.322
κ_{B_2}	0.571	0.914	0.548	0.391	0.330
κ_{C_1}	0.273	0.419	0.855	0.189	0.354
κ_{C_2}	0.279	0.417	0.847	0.197	0.354
κ_{D_1}	0.295	0.179	0.247	0.446	0.293
κ_{D_2}	0.296	0.176	0.245	0.447	0.289
κ_{E_1}	0.263	0.336	0.364	0.194	0.553
κ_{E_2}	0.256	0.335	0.362	0.198	0.552

TABLE 6.2. PERCENTAGE OF TOPK RANKINGS LABELED PER USER

	A	B	C	D	E
κ_{A_1}	52.5	24.2	18.6	3.1	1.6
κ_{A_2}	54.6	21.9	19.0	3.6	0.9
κ_{B_1}	1.0	97.8	0.5	0.0	0.7
κ_{B_2}	1.6	96.2	1.4	0.0	0.8
κ_{C_1}	0.0	0.0	99.5	0.0	0.5
κ_{C_2}	0.0	0.0	99.4	0.0	0.4
κ_{D_1}	6.0	4.1	2.8	82.7	4.4
κ_{D_2}	6.4	4.1	3.6	83.1	2.8
κ_{E_1}	3.9	7.4	15.0	1.3	72.4
κ_{E_2}	2.2	6.6	15.6	1.3	74.3

7. Algorithm to Calculate Overlapping TopK Rankings from Network Traffic

Network-user profiling has been used to detect unknown attacks by trying to identify an unexpected network behavior, which is a common symptom of security problems. [64, 70, 82]

User profiling can be achieved by analyzing host traffic, building TopK ranking (a list containing the k top elements of the ranking) of reached services, and using ranking similarity measures [60]. Host traffic includes a few headers of every network packet. Nevertheless, a single user can produce gigabytes of data during one labor week.

The algorithm introduced in [60] is designed to run on only one host because of its sequential nature. An important limitation of the algorithm is that it requires to keep in memory a huge amount of data in order to build all the TopK rankings.

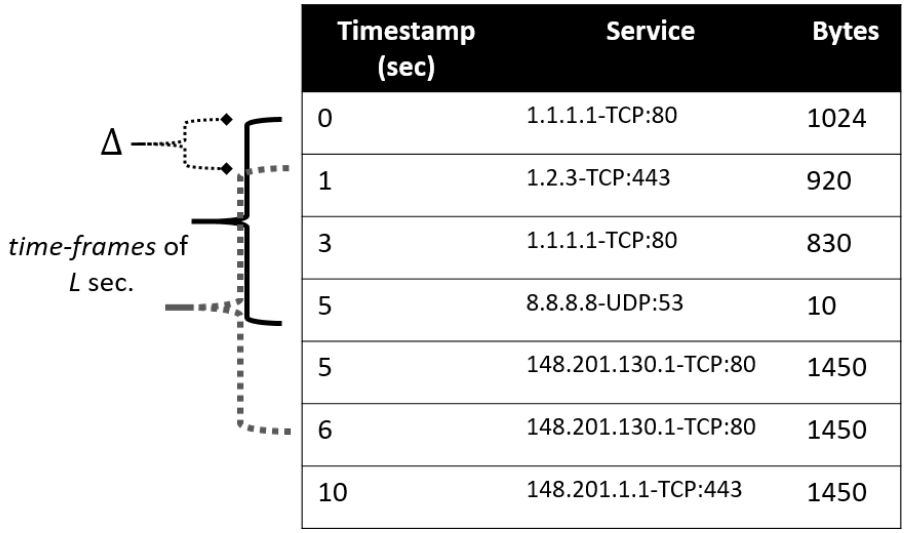
In this chapter we present a full complexity analysis of such algorithm with the purpose of knowing its limitations.

7.1. Algorithm Description

As mentioned in [60], the captured traffic is arranged by time-frames. A time-frame contains all the packets captured in a specific period L. The staring points of every time-frame are separated by Δ seconds. Δ is smaller than L, thus time-frames are overlapped. This arrangement is represented in Fig. 7.1. A TopK ranking of reached services is calculated for each time-frame, using the total bytes transferred as weight criterion.

In general, Algorithm 1 reads the input data from a CSV file, where each line represents a network packet, and produces a CSV file where each line contains a TopK ranking. For each line from $InputFile$, a $Packet$ defined as $\{ timestmap, service, bytes\}$ is built and added to the list P. Then, P is sorted by $timestamp$ in ascending order.

The starting point t_s of the first time-frame is calculated as the greatest multiple of Δ not greater than the $timestamp$ of the first packet. All the packets from P that belongs to the first time-frame, i.e. the timestamp is between t_s and $(t_s + L)$, are selected. After this, the total amount of bytes



Timestamp (sec)	Service	Bytes
0	1.1.1.1-TCP:80	1024
1	1.2.3-TCP:443	920
3	1.1.1.1-TCP:80	830
5	8.8.8.8-UDP:53	10
5	148.201.130.1-TCP:80	1450
6	148.201.130.1-TCP:80	1450
10	148.201.1.1-TCP:443	1450

Figure 7.1: Overlap $time\text{-}frame$ representation with: $L = 5$ sec and $\Delta = 1$ sec.

transferred from each reached service is calculated and stored in T as a pair $< service, bytes >$. Then, T is sorted by $bytes$ in descending order. From the first K services, a new TopK ranking is built, and added to the output.

The process repeats with a new starting point: $t_s + \Delta$, and continues until the starting point reaches the last packet.

7.2. A priori Analysis

7.2.1 Time Complexity

The first step of the algorithm reads the input file line by line. From each line, it extracts key information, hereafter denoted as $Packet$, and we add it to a list for further processing. The second step sorts the list in ascending order by time-stamp. Equation (7.1) expresses the time required by these two steps, where n is the number of lines in the file and p denotes the time required to parse a line and build a $Packet$ object.

$$T_1(n) = np + nlog(n) \tag{7.1}$$

The next step of the algorithm is the generation of TopK rankings. Since the time required by that this operation depends on the number of packets contained in each time-frame, this analysis assumes the number of packets contained in a time-frame is upper-bounded by m. This step builds a map

Algorithm 1 TopK ranking Generation

$L \leftarrow$ Time-frame length
$\Delta \leftarrow$ Time between each time-frame start point.
$K \leftarrow$ Number of elements in each TopK ranking.
$Packet = \{timestmap, service, bytes\}$
$P \leftarrow$ an initially empty list of $Packet$.
for each $line$ **in** $InputFile$ **do**
    $p \leftarrow Packet$ built from $line$
    $P \leftarrow P \cup p$
end for
sort P by $timestamp$ in ascending order
$t_s \leftarrow \lfloor \frac{P[1].timestamp}{\Delta} \rfloor * \Delta$
$j \leftarrow 1$
while $j \leq |P|$ **do**
    $i \leftarrow j$
    $t_e \leftarrow t_s + L$
    $T \leftarrow$ an empty map of pairs $< service, bytes >$
    while $P[i].timestamp < t_e$ **do**
        if $P[i].timestamp < t_s$ **then**
            $j \leftarrow j + 1$
            $i \leftarrow i + 1$
            continue
        end if
        if $< P[i].service, b > \in T$ **then**
            $b \leftarrow b + P[i].bytes$
        else
            $T \leftarrow T \cup < P[i].service, P[i].bytes >$
        end if
        $i \leftarrow i + 1$
    end while
    sort T by $bytes$ in descending order
    $line \leftarrow$ an empty string
    for $k \leftarrow 1$ **to** K **do**
        $< s, b > \leftarrow T[k]$
        $line \leftarrow$ CONCAT$(line, s)$
    end for
    WRITE$(Output, line)$
    $t_s \leftarrow t_s + \Delta$
end while

<service, bytes> on each time-frame. The number of get operations to the map is m because each packet from the time-frame is searched into the map. A get operation takes $O(lg(m))$; therefore, filling the map takes $O(mlg(m))$. After all the packets from the current time-frame are processed, the map is sorted in descending order by values, and the first k elements are selected. Thus, the equation that expresses the time complexity of this step is the following:

$$2mlg(m) + k \tag{7.2}$$

After each time-frame is processed and the TopK ranking built, the algorithm searches the starting point of the next time-frame. This analysis assumes the number of packets in Δ is upper-bounded by d, i.e., the distance in packets between starting points of two consecutive time-frames is not greater than d. This step adds d to the previous equation:

$$2mlg(m) + k + d \tag{7.3}$$

This TopK ranking generation is executed once for each time-frame. The total number of time-frames can be calculated as: $\frac{n}{d}$. Thus, the time required to generate all the TopK rankings is expressed as:

$$T_2(n) = \left(\frac{n}{d}\right)(2mlg(m) + k + d) \tag{7.4}$$

Finally, the time complexity of the entire algorithm can be expressed as the sum of equations 7.1 and 7.4:

$$T(n) = T_1(n) + T_2(n) = np + nlog(n) + \left(\frac{n}{d}\right)(2mlg(m) + k + d) \tag{7.5}$$

with simple algebra, it can be rewritten as:

$$T(n) = n\left(p + log(n) + \frac{2mlg(m) + k}{d} + 1\right) \tag{7.6}$$

Considering that m, d and k are constants much smaller than n (hundreds of thousands times), from Equation 7.6 we can see that the time complexity of the algorithm is linear with respect to the number of packets, i.e., $T(n) \in O(n)$.

7.2.2 Space Complexity

During the first step of the algorithm, only the current line read is stored in memory; from this line a $Packet$ object is built and added to a list. Equation 7.7 expresses the memory required by the first step, where l represents the size of the line, p the size of a $Packet$ and n is the number of lines in the input.

$$M_1(n) = l + np \tag{7.7}$$

In the next step, TopK ranking generation, a map is built. On each time-frame, this map is first emptied and then filled with at most m <service, bytes> entries of size s each. At the end, a TopK ranking of services is created and added to a global collection. Thus, the equation that expresses the memory used by this step is the following:

$$M_2(n) = ms + \left(\frac{n}{d}\right) k \tag{7.8}$$

Therefore, the time complexity of the algorithm is:

$$M(n) = M_1(n) + M_2(n) = l + np + ms + \left(\frac{n}{d}\right) k \tag{7.9}$$

with simple algebra, it can be rewritten as:

$$M(n) = n\left(p + \frac{k}{d}\right) + l + ms \tag{7.10}$$

Considering, as in the time complexity, that l, m, d and k are constants much smaller than n, from Equation 7.10 we can see that the space complexity of the algorithm is linear with respect to the number of packets, i.e., $M(n) \in O(n)$.

7.3. A posteriori Analysis

The algorithm was implemented in Java 8. Nine datasets with contrasting number of packets were employed in this analysis [see Table 7.1]. All the executions were carried out in an Intel Xeon E5 CPU with four cores @ 2.4 GHz, 16 GB RAM and with a clean installation of Ubuntu Server 16.04.

TABLE 7.1. DATASET CHARACTERISTICS

Dataset	Size (KB)	Packets
1	4,069,004	30,392,821
2	2,151,220	15,852,504
3	1,105,876	8,205,236
4	613,892	4,564,003
5	347,416	2,575,992
6	168,168	1,250,000
7	34,716	257,599
8	17,356	129,000
9	3,508	25,759

The program was executed four times for every dataset, varying the heap size: 14 GB, 6 GB, 5 GB and 4 GB. Note that the second and third heap sizes are barely higher than the largest dataset.

Using 14 GB, we obtained a linear relation between the number of packets and the execution time. This behavior is shown in Fig 7.2a where three sets of data are plotted: 1) the time spent loading data into memory, 2) the time used to generate all TopK rankings, and 3) the total time. The plot includes a dotted line that represents the linear equation: $f(x) = 0.0139x + 817.51$, that best fits the total time.

Concerning to second and third executions (6 and 5 GB heap sizes), we can observe that the total time required by the largest data set is much higher than the linear behavior expected [see Fig. 7.2b and 7.2c]. The extra time is spent by the garbage collector in allocating and releasing memory.

The fourth execution (4 GB heap size) resulted in a "GC overhead limit exceeded" error when processing the largest dataset; this means that the garbage collector was unable to recover more than 2% of the heap. The rest of the datasets behaved as expected: linear execution time [see Fig 7.2d].

7.4. Conclusions

In this work, we have analyzed the algorithm that generates TopK rankings, first introduced in [60]. From this analysis we can conclude the following: 1) the time complexity is linear with

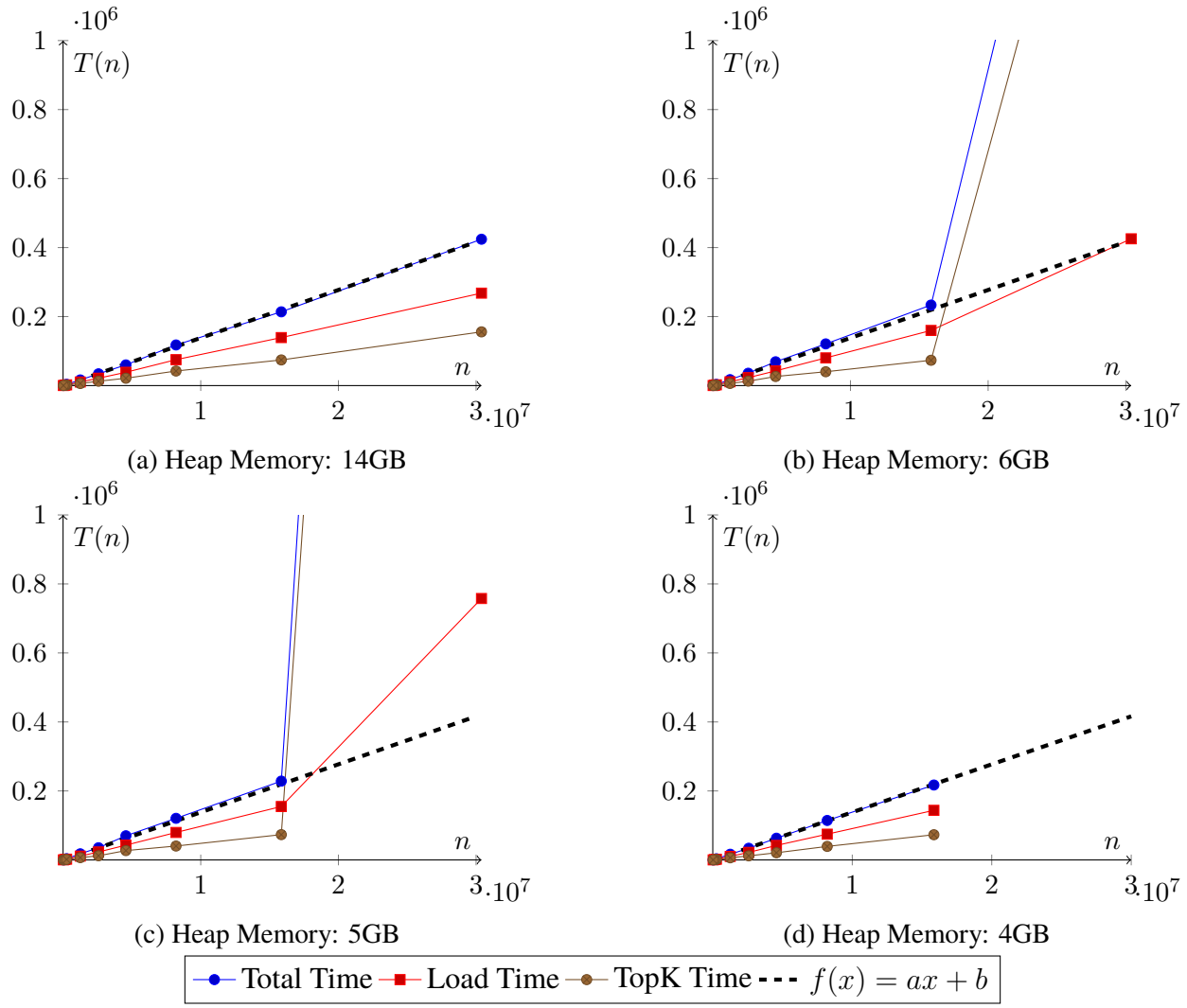


Figure 7.2: Relation between execution time and number of elements.

respect to the number of packets, 2) the space complexity is also linear with respect to the number of packets, 3) the first conclusion holds only if the heap size is big enough to process all the data.

As we can see in the charts, as the program manages to allocate more memory for storing new packet data, the efficiency of the algorithm degrades dramatically to the point of ending abruptly when available memory is not enough.

In order to address memory issues, the change of the programming paradigm from sequential to distributed is suggested.

8. MapReduce approach to Build Network User Profiles with TopK Rankings

Network-user profiling has been used as a security technique to detect unknown or malicious behaviors [64, 70, 82].

The algorithm introduced in [60] and analyzed in Chapter 7 is designed to run in a single thread. The main problem of this algorithm is the huge amount of memory it requires resulting eventually in an abrupt termination when available memory is not enough due the memory management module from the operating system, as a protection.

MapReduce is a programming model and an associated implementation for processing and generating large data sets [24]. The model consists on specifying: a) a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and b) a $reduce$ function that merges all the intermediate values. Programs written in MapReduce are naturally parallelizable and can be executed on a large cluster of commodity machines.

In this chapter, we propose a MapReduce program that generates the TopK rankings as defined in [60]. This program has the following benefits with respect to the algorithm presented in Chapter 7:

a) Addresses memory allocations and out-of-memory errors.

b) Distributes workload among various workers.

8.1. MapReduce Concepts

MapReduce programming model was proposed by Dean and Ghemawat from Google, as a model for processing and generating large data sets. The programs under this model are naturally parallelizable and can be executed in large clusters of machines [24].

The model consists on specifying: a) a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and b) a $reduce$ function that merges all the intermediate values.

The overall flow of a MapReduce operation is illustrated in Fig. 8.1

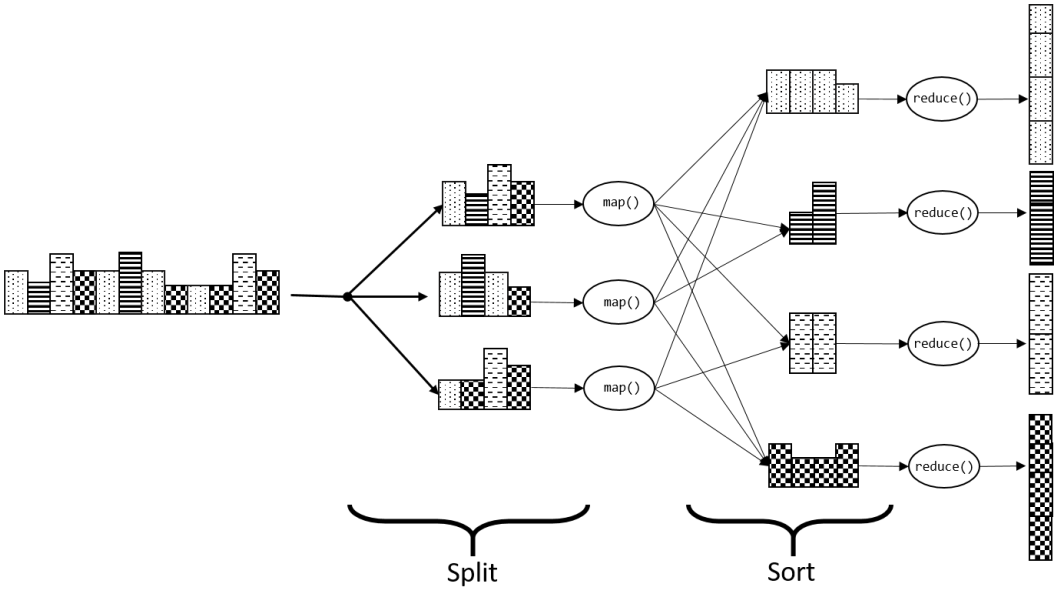


Figure 8.1: Overall flow of a Map Reduce Operation

8.2. MapReduce Algorithm

Since the TopK generation algorithm has no functional dependencies, a MapReduce version of this algorithm is feasible. The execution environment is expected to distribute the workload among various workers, and to address all the memory allocation issues, no matter the size of normal-behavior datasets.

As we can see in Fig. 8.2, the *map* function associates each packet with all the time-frames it belongs to. Each time-frame is identified by its lower bound. The *reduce* function turns the packet list of a time-frame into a TopK ranking.

8.2.1 Map function

The Map function, described in Algorithm 2, receives a line (of the input dataset) which contains data of a single packet, and generates a 2-tuple $< id, P >$ for each time-frame the packet belongs to.

At the beginning, the initial $\frac{L}{\Delta}$ packets have no time-frame to be assigned because we have not yet defined time-frames at the start of the process; they will be tagged as incomplete.

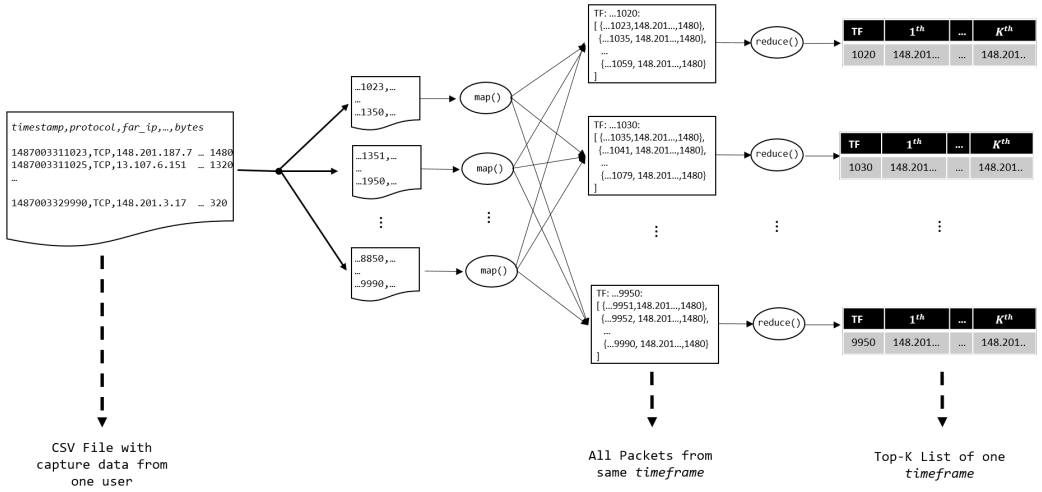


Figure 8.2: Flow diagram of the proposed MapReduce algorithm

Algorithm 2 Map Function

```
1: function MAP(line)
2:     p ← Packet built from line
3:     t ← ⌊P.timestamp/Δ⌋ * Δ
4:     while t ≥ 0 AND p.timestamp < (t + L) do
5:         WRITE(t,p)
6:         t ← t − Δ
7:     end while
8: end function
```

8.2.2 Reduce function

The Reduce function, described in Algorithm 3, receives a list containing all pf the packets belonging to a specific time-frame. From this list, the total bytes transferred for each reached service is calculated, and stored into a map T that associates different $Service$ with the $TotalBytes$ transferred. Then T is sorted by its value in non-increasing order. The first K reached services are selected as TopK. Finally, the TopK is written to the output as a comma-separated list.

8.3. Analysis of MapReduce Approach

This approach comprises only two functions: *map* and *reduce*. Both functions run in a distributed environment, so the analysis of the algorithm is carried out for each function assuming an evenly distributed workload among w workers.

Algorithm 3 Reduce Function

```
1: line ← an empty string
2: function REDUCE(key,P[ ])
3:     for all p ∈ P do
4:         if < p.service, b >∈ T then
5:             b ← b + p.bytes
6:         else
7:             T ← T∪ < p.service, p.bytes >
8:         end if
9:     end for
10:    sort T by bytes in descending order
11:    for k ← 1 to K do
12:        < s, b >← T[k]
13:        line ← CONCAT(line, s)
14:    end for
15:    WRITE(key,line)
16: end function
```

8.3.1 Time Complexity

The *map* function parses the input line into a $Packet$ object, and generates a duple <time-frame, packet> for each time-frame containing this packet. Equation (8.1) expresses the total time required to process the n lines of the input file, where p denotes the time required to parse a line and build a $Packet$ object, and s represents the time required to write the tuple.

$$T_{map}(n) = \frac{n}{w}\left(p + \frac{L}{\Delta}s\right) \tag{8.1}$$

The total time required by *reduce* function to build a TopK ranking for each list of packets is expressed in (8.2), where d represents the average number of packets in Δ, m is the average number of packets in a time-frame, and s represents the time required to write the output.

$$T_{reduce}(n) = \frac{n}{wd}\left(2mlgm + k + s\right) \tag{8.2}$$

Finally, the time complexity of the entire algorithm can be expressed as the sum of both equations.

$$T(n) = \frac{n}{w}\left(p + \frac{L}{\Delta}s + \frac{2mlgm + k + s}{d}\right) \tag{8.3}$$

As in the original algorithm, considering that m, d, s and k are constants much smaller than n, from (8.3) we can see that the time complexity of the algorithm is also linear with respect to the number of packets, i.e., $T(n) \in O(n)$.

8.3.2 Space Complexity

A more precise analysis of the space complexity not only depends on the algorithms, but also on the MapReduce framework. However, this analysis considers only the memory used by map and $reduce$ functions.

The map function stores in memory only the current line(l) and generates a $Packet$ object (p). Equation (8.4) expresses the memory required by *map* function. We assume that w instances of map function are executed simultaneously.

$$M_{map}(n) = w(l + p) \tag{8.4}$$

The *reduce* function receives a list of m packets. It builds a map with at most $m < service, bytes >$ entries of size s each. At the end, a TopK ranking of services is created and written to the output. Thus, the equation that expresses the memory used by w simultaneous executions of *reduce* is:

$$M_{reduce}(n) = w(ms + k + l) \tag{8.5}$$

From equations (8.4) and (8.5), we can observe that the space complexity of the program is independent of the number of packets. Therefore, it is constant with respect to this number, i.e, $M(n) \in O(1)$.

8.4. Hadoop

Hadoop is an open-source project self defined as: "A framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models."[1]

Hadoop offers two different execution modes for MapReduce solutions: 1) the single-node mode, and 2) the cluster mode. In the former, all the modules of the framework are executed at the same

[1]"http://hadoop.apache.org(Visited: Aug 01, 2018)"

host. This mode is recommended for testing purposes.

Hadoop supports Java as its default language, thus MapReduce programs are expected to be built as a JAR file. In order to run a MapReduce program using Hadoop we need three things: a map function, a reduce function and a job[62].

The map function must be implemented in a class that extends the abstract class $Mapper$ which defines the abstract method $map()$. This method receives the pair $< key, value >$ to be processed, and invokes $write$ method to send the intermediate pair(s) $< key, value >$ that will be processed on the next phase.

Similarly, the reduce function must be implemented in a subclass of class $Reducer$ which defines the abstract method $reduce()$. This method receives a single key and a collection containing all the $value$ associated with the key. The $reduce$ method invokes the $write$ method to send the final par <key, $value$> to the output.

Finally, a Hadoop program requires a Job object which encapsulates the specification of the program: the mapper class, the reduce class, input and output definitions, among other parameters.

8.5. AWS & EMR

Amazon Web Services (AWS) is the cloud service provider of Amazon. AWS was officially reveled to the world on 2006 with a single service of storage (S3)[32]. Nowadays, it offers over one hundred services. Some examples of such services include the following:

- EC2, virtual servers.
- S3, storage.
- Route53, scalable DNS.
- Lambda, serverless context.
- RDS, relational databases.
- DynamoDB, NoSQL databases.
- Amazon SageMaker, Machine Learning models.

- EMR, Map Reduce clusters.

Amazon Elastic Cloud Computing, EC2, provides a virtual machine that follows a configuration selected by the user, which includes: number of CPUs, amount of RAM, storage, operating system and network setup.

Amazon Elastic Map Reduce, EMR, is a managed service that provides big data analytics frameworks such as: Apache Hadoop, Apache Spark, Apache HBase and Presto straight out the box and ready for use[78].

EMR is organized in clusters. A cluster is a group of EC2 instances, i.e. virtual machines, running the same framework simultaneously. Each cluster can have associated an auto-scaling policy to increase/decrease the number of EC2 instances.

The general process for using the EMR service is the following: 1) Choose the cluster configuration 2) Deploy the cluster 3) Upload program and data to S3, the file storage of AWS. 4) Create a task, which includes: a) the program to run, b) the input and c) the path in S3 where the output will be stored.

8.6. Implementation

Our implementation follows the basic form of a MapReduce Hadoop program. We use Java because it is the native language for Hadoop and also is the supported format of AWS for Hadoop cluster.

The Hadoop program consists of three Java classes: 1) A main class containing the Job definition and the $main$ function, 2) the Mapper class and 3) the Reducer class.

The main class called TopKMapReduce contains the main method where the program is configured (see Algorithm 4). Lines 8 and 9 specify the Mapper and the Reducer classes, respectively. Since the mapper produces a pair <timestamp, packet>, the datatypes of the intermediate key and value are set to LongWritable and PacketWritable, respectively (see lines 10 and 11). The reducer produces a pair <timestamp, TopK>.

The map function (see Algorithm 5) receives a line of the input dataset that contains data of a single packet, and produces a pair $< id, P >$ for each different time-frame id that the packet p belongs to.

The *reduce* function (see Algorithm 6) receives a list containing all the packets that belong to a specific time-frame. It creates a map that associates each reached service <IP, protocol, port> with the total number of bytes transferred from/to it. Then, the map is sorted by reached service in nonincreasing order. The first K reached services from the sorted map are selected as TopK. Finally, the TopK is written to the output as a comma-separated list.

Algorithm 4 TopKMapReduce.java

```
1 public class TopKMapReduce {
2     public static void main(String[] args) throws IOException,
3             InterruptedException, ClassNotFoundException {
4
5         Configuration conf = new Configuration();
6         Job job = Job.getInstance(conf, "TopK Builder");
7         job.setJarByClass(TopKMapReduce.class);
8         job.setMapperClass(TopKMapper.class);
9         job.setReducerClass(TopKReducer.class);
10        job.setMapOutputKeyClass(LongWritable.class);
11        job.setMapOutputValueClass(PacketWritable.class);
12        job.setOutputKeyClass(Text.class);
13        job.setOutputValueClass(Text.class);
14
15        FileInputFormat.addInputPath(job, new Path(args[0]));
16        FileOutputFormat.setOutputPath(job, new Path(args[1]));
17
18        int exitvalue = (job.waitForCompletion(true)) ? 0 : 1;
19
20        System.exit(exitvalue);
21    }
22 }
```

8.7. Experiments and Results

8.7.1 First Experiment: Precision test

In order to validate that the MapReduce approach is correct, i.e., produces the same result than the original algorithm, a ∼4GB dataset containing at least thirty million packets was processed using both algorithms, and the results were compared.

The sequential algorithm produced 18,372 TopK lists whereas the MapReduce algorithm produced 18,401 TopK lists. The extra 29 lists are because the intermediate key generation method associates the first packets to non-existent timeframes. If we remove these first lines both outputs will be exactly the same.

Algorithm 5 TopKMapper.java

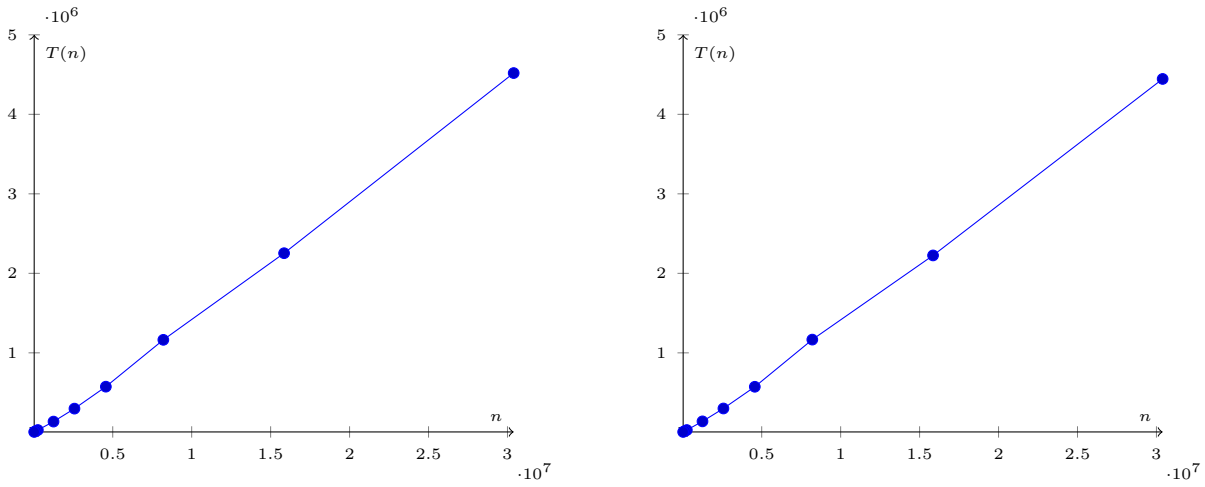
```
1 public class TopKMapper extends Mapper<Object, Text,
2                                        LongWritable, PacketWritable> {
3
4     protected void map(Object key, Text value, Context context) throws
          IOException, InterruptedException {
5
6         String line = new String(value.toString());
7         Packet p = Packet.buildPacket(line);
8         if(p == null) { return; }
9         long ts_key  = (p.getTs_packet() / DELTA_TIME)*DELTA_TIME;
10        while(ts_key >= 0 && p.getTs_packet() < (ts_key + TIME_FRAME_SIZE)) {
11            LongWritable send_key = new LongWritable(ts_key);
12            context.write(send_key, new PacketWritable(p));
13            ts_key = (ts_key - DELTA_TIME);
14        }
15    }
16
17 }
```

8.7.2 Second Experiment: Performance test

The purpose of this experiment is to prove that the program efficiency is not affected by the lack of RAM memory as happens in the original algorithm. All the datasets mentioned in Table 7.1 were processed using Hadoop in standalone mode running into a AWS instance type T2.xlarge which has 4 vCPU and 16GB of RAM. Two heap memory sizes: 1GB and 10GB was configured. Fig 8.3 shows the total time employed by each dataset with both configurations. We can see that they produce similar plots, confirming that the out-of-memory problem is solved by the MapReduce framework.

8.7.3 Third Experiment: Scalability test

This experiment was carried out in Amazon EMR cluster. Datasets 1 and 2 were processed ten times, increasing the number of task nodes (workers). The total time used to process the data was registered and plotted. Fig. 8.4 shows these times, and includes a dotted line depicting the time required by the original algorithm running with enough memory (14 GB) to remain efficient. We can see that using eight task nodes, the execution time is improved by the MapReduce algorithm. Equation (8.6) shows an upper bound of the execution time of the MapReduce algorithm in terms of



(a) Heap Memory: 10GB

(b) Heap Memory: 1GB

Figure 8.3: Execution times with Hadoop in Standalone mode, using different amount of packets and two heap memory sizes

the number of workers.

$$t(w) = \frac{t(1)}{\sqrt{w}} \tag{8.6}$$

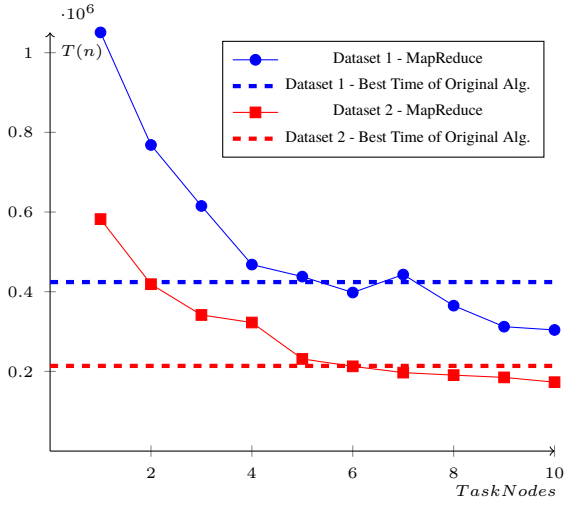


Figure 8.4: Execution times of the MapReduce algorithm in an Amazon EMR Cluster with Hadoop

8.8. Conclusions

In this chapter, we have presented a MapReduce approach of an algorithm that profiles network users by means of TopK rankings of reached services, which was introduced in [60]. The primary motivation behind this implementation is to prevent out-of-memory errors that arise when the algorithm processes huge datasets. Another motivation is to take advantage of the parallel nature of MapReduce approach

From the experiments carried out, we conclude the following concerning the MapReduce algorithm: 1) it produces the same output as the original algorithm, 2) its efficiency does not depend on the amount of available RAM memory, and 3) it is scalable, i.e. as the number of employed task nodes grows, the execution time decreases as equation 8.6 represents.

Algorithm 6 TopKReducer.java

```
1 public class TopKReducer extends Reducer<LongWritable, PacketWritable, Text,
    Text> {
2
3     protected void reduce(LongWritable key, Iterable<PacketWritable> values,
      Context context)
4             throws IOException, InterruptedException {
5
6         String k = key.toString();
7         HashMap<String, Long> ips = new HashMap<>();
8         for (PacketWritable p : values) {
9             String packetID = p.getFullDst();
10            long bytes = p.getBytes();
11            if (ips.containsKey(packetID)) {
12                long totalBytes = ips.get(packetID);
13                ips.put(packetID, totalBytes + bytes);
14            } else {
15                ips.put(packetID, bytes);
16            }
17        }
18
19        List<Map.Entry<String, Long>> listEntrys = new ArrayList<>(ips.
              entrySet());
20        listEntrys.sort((Map.Entry<String, Long> o1, Map.Entry<String, Long>
              o2) -> {
21            return o2.getValue().compareTo(o1.getValue());
22        });
23
24        //Trunk the list to k elements.
25        ArrayList<String> topK_elements = new ArrayList<>();
26        int iMax = (listEntrys.size() < K_SIZE) ? listEntrys.size() : K_SIZE;
27        for (int i = 0; i < iMax; i++) {
28            topK_elements.add(listEntrys.get(i).getKey());
29        }
30        String out = "";
31        boolean first = true;
32        for(String ip : topK_elements) {
33            if(first) {
34                out+=ip;
35                first = false;
36            } else {
37                out+="\t"+ip;
38            }
39        }
40        context.write(new Text(k), new Text(out));
41    }
42 }
```

9. A Serverless approach to Build and Evaluate Network User Profiles with TopK Rankings

In cybersecurity, the use of profiles to detect anomalous behaviors has been a common technique in different research works [64, 70, 82]. In [60], we proposed a new methodology for building user profiles based on calculating TopK rankings of reached services. Experimental results show that the proposed method is capable to detect unexpected behaviors of networks users, which are commonly caused by either trojans, virus or intrusions. The authors in [59] presents the implementation of such methodology using the traditional sequential programming, and the MapReduce programming model.

Both sequential and MapReduce programming models, presented in [59], require an always-on infrastructure, which represents high direct costs and some indirect costs, including infrastructure support. A much more affordable and yet efficient technology to implement our methodology is serverless computing. Unlike other cloud technologies, as virtual machines on demand, serverless computing produce charges only when it is used, i.e., when our cloud application is running.

Serverless computing -or function as a service- is a new paradigm for the deployment of cloud applications [73] where the developer is not concerned on the infrastructure where their code runs, but only on writing good code. The cloud-service provider charges the user according to the time and memory consumed by their program.

In this chapter, we propose an execution architecture for the proposed methodology using a serverless model.

9.1. TopK Network Profiles and Security

The methodology proposed in [60] has two big phases: 1) a learning phase, where the user-network profile is built, and 2) the evaluation phase, where the real-time traffic of a user is compared to their profile traffic in order to detect some anomalous behavior.

The profile is a collection of TopK rankings, such that each TopK ranking is a non-increasing list containing ten reached services with the highest amount of transferred bytes during a period of

five minutes.

During the learning phase, a TopK ranking is built every ten seconds using the network traffic captured during the last five minutes; then, it is stored into an unique-values dataset. The TopK rankings are overlapped.

During the evaluation phase, a TopK ranking is calculated every ten seconds using the network traffic captured during the last five minutes; then, it is compared with every TopK belonging to the user's profile. This comparison is performed using Average Overlap measuring method which calculates how similar two undefined TopK rankings are. This process is illustrated in Fig. 9.1.

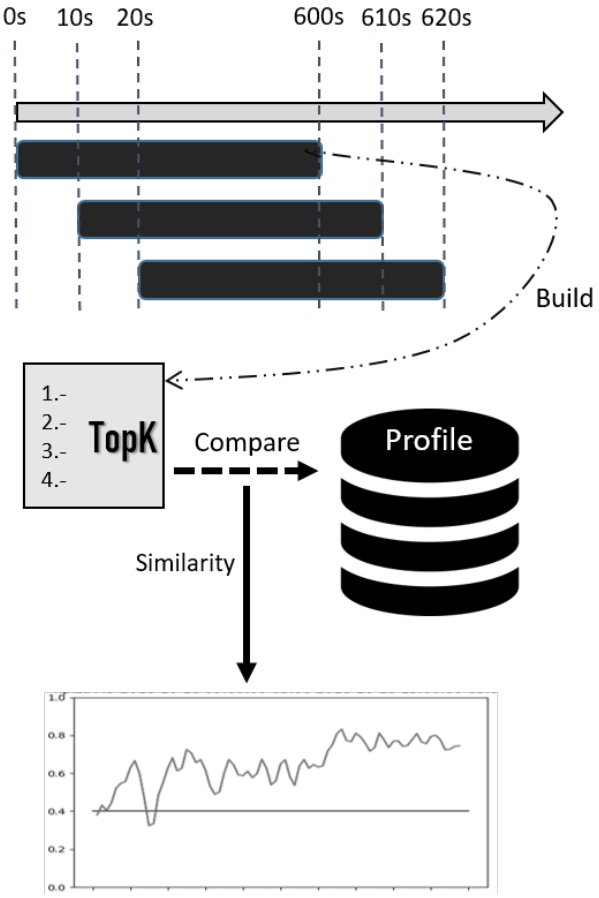


Figure 9.1: Real-time traffic evaluation process

9.2. Serverless Architecture

Serverless -or Function as a Service (FaaS)- has emerged as a new paradigm for the deployment of applications and services where the logic of the application is split into functions that execute

as response of events. [73]. The main cloud providers that offer serverless computing include the following: AWS Lambda[15]; Google Cloud Functions[33], Azure Functions[51], IBM Cloud Functions[40].

The serverless architecture is also an event-driven architecture because each function is invoked by either a specific event or by group of events. A primary advantage of this architecture is the possibility to execute a virtually unlimited number of instances of a function.

In a serverless environment, functions have limited time and memory resources. Therefore, they need to be lightweight, scalable and single-purpose. Some good practices for designing these functions include the following: a) keep the code small and efficient, b) a long function should be split into a sequence of smaller functions, c) each function should process only the event data, i.e., they should not need to call another function to fully accomplish its purpose.

Under the 'As-a-Service' logic, the user pays only for the amount of resources that were employed during the execution of all the functions, including: CPU time, memory and bandwidth.

The serverless architecture is adequate to implement the mentioned methodology because it can be easily divided into many smaller steps such that each runs independently and processes different pieces of data. The benefit of this architecture is the possibility to process all the information directly in the cloud without requiring an "always-on" infrastructure.

9.3. AWS Services

AWS is the cloud-service provider of Amazon. AWS started offering IT infrastructure services in 2006 with a simple storage service; nowadays, it offers over a hundred different cloud services [4] and is the world's most broadly adopted cloud platform [80].

9.3.1 AWS S3

Amazon Simple Storage Service (S3) is an object storage service that offers scability, availability and security. It offers an eleven 9s(99.9999999%) of durability of the stored information, and a 99.99 of availability [10].

9.3.2 AWS SQS

Amazon Simple Queue Service (SQS) is a managed message-queue service. SQS can send, store, and receive any volume of messages. SQS supports two different types of queues: 1) the standard queue, which provides best-effort ensuring that messages are generally delivered in the same order as they're sent, and 2) the FIFO queue, which guarantees that the order is preserved [11].

9.3.3 AWS Lambda

AWS Lambda is a serverless compute service that runs code in response to events. It offers a high-availability auto-scaling context to execute code. The code can be written in any of the languages that AWS Lambda supports, which include (but are not limited to) the following: Node.js, Java, C#, Go and Python.

9.3.4 AWS DynamoDB

Amazon DynamoDB is a NoSQL database that supports key-value and document data models. It can support virtually any size tables with horizontal scaling. The tables can be automatically replicated across different regions of AWS; also, DynamoDB can auto scale, up and down, by monitoring the performance usage of the applications [9].

9.3.5 AWS API Gateway

Amazon API Gateway is a fully managed service to publish, maintain, monitor and operate API at any scale. It supports HTTP/Rest APIs and WebSocket APIs [8].

9.4. Proposed Architecture

The proposed architecture, depicted in Fig. 9.2, is constituted by four main blocks, which are explained in detail in the following subsections.

The architecture receives one or many files containing each a list of TopK rankings, where each row of the file contains one TopK. These files are generated by a *Capture Application*(1), which runs at the user machine. This application has three primary functions: 1) capture all the network

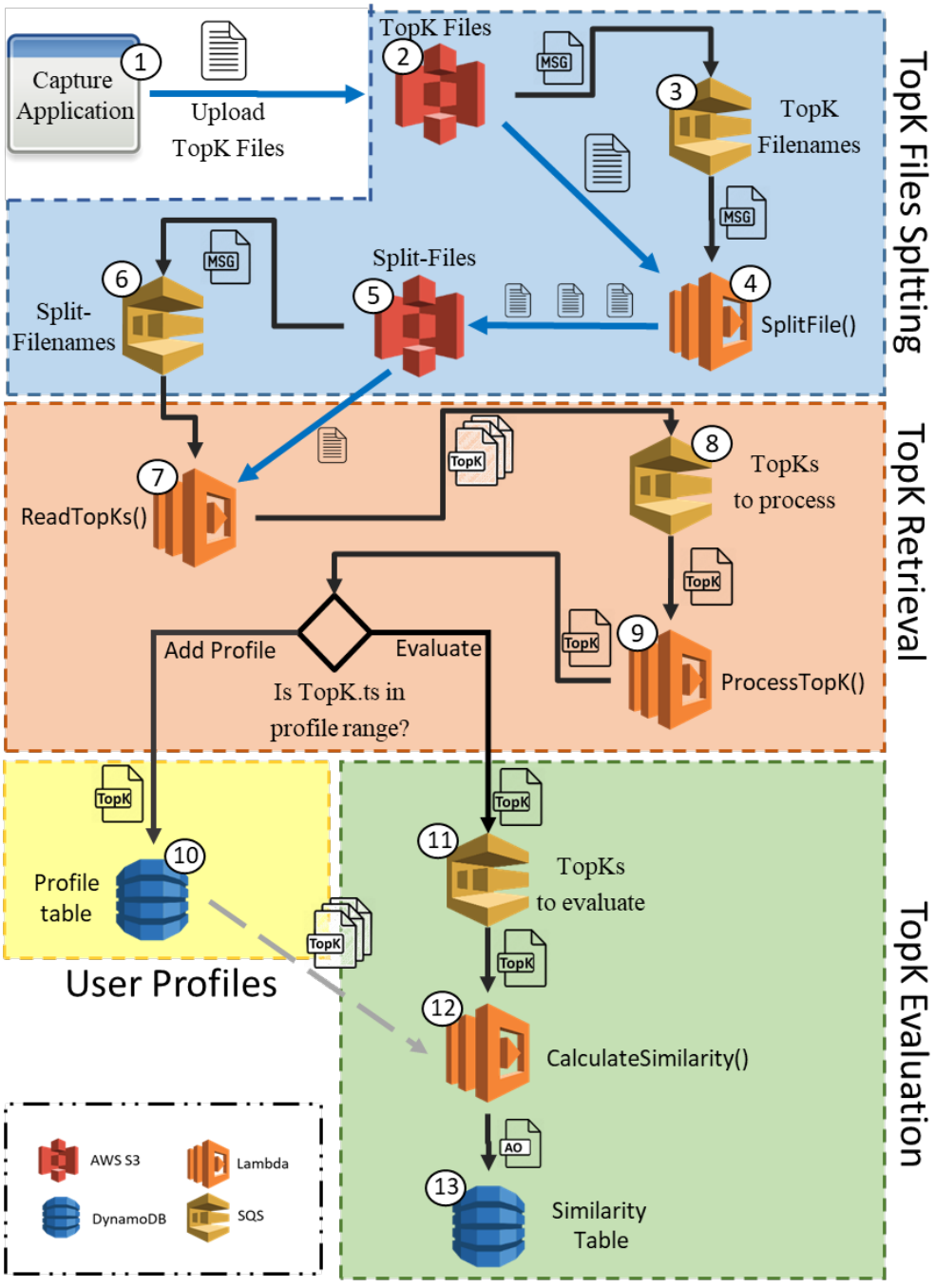


Figure 9.2: Architecture of the proposed serverless implementation

traffic in the user host, 2) calculate the TopK rankings of reached services, and 3) upload the file(s) containing the TopK rankings to AWS S3.

9.4.1 TopK File Splitting

The purpose of this block is to partition all the TopK files -uploaded by the Capture Application- in a set of much smaller TopK files, in order to ensure that the functions that process TopK data finish in a short time.

Each time a file or set of files is uploaded to the *TopK Files*(2) bucket, a message containing the names of the uploaded files is sent to the *TopK Filenames*(3) queue. Function *SplitFile*(4) is triggered by a message received from *TopK Filenames* queue (3). This message contains the filename of the TopK file to split. The function reads the TopK file from S3 and splits it into smaller files. Each split file is uploaded to S3 into *Split-Files* bucket (5). This queue guarantees that every filename will be processed by the function sooner or later, thus, every TopK file will be partitioned into small split files.

Each time a split file is added to *Split-Files* bucket (5), a message containing the filename is added to *Split-Filenames* queue (6).

9.4.2 TopK Retrieval

This block has two primary functions: 1) load each TopK from every split file produced in the previous block, and 2) decide whether a TopK should be added to the user profile or should be evaluated.

The function *ReadTopKs*(7) is triggered by *Split-Filenames* queue (6). This function loads the appropriate file from S3, and adds each TopK to *TopKs-to-process* queue (8).

ProcessTopK function (9) receives each TopK and reads its timestamp. According to this value, the TopK is either inserted into a *Profile* table (10) or added to a queue containing TopKs to evaluate (11).

9.4.3 User Profiles

The user profiles are stored in a AWS DynamoDB table. Each item consists of only a key attribute that stores a TopK as a string value. This way, this table cannot have two or more items with the same TopK.

When *ProcessTopK*(9) decides that a TopK belongs to the user profile, this function inserts the TopK into table *Profile*(10).

9.4.4 TopK Evaluation

This block evaluates each TopK received according to the user profile, by calculating a similarity factor.

TopKs to evaluate(11) is filled with every TopK to be evaluated. *CalculateSimilarity*(12) function reads one TopK from the queue and calculates its similarity factor against each TopK of the profile. The greatest factor is inserted into *Similarity*(13) table.

9.5. Architecture Implementation

The Capture Application was developed using Java 8 and the Pcap4J library[1] for packet-capture functions. It was installed as an auto-start service. The rest of the architecture was implemented using AWS services under the serverless logic.

Table 9.1 shows the global configuration used for the AWS services employed. Every S3 bucket has been configured to respond to upload-file events (PUT, POST). Table 9.2 shows the notification destination of each bucket.

All the AWS Lambda functions have been configured with a SQS Queue as trigger. For each function, Table 9.3 shows: a) the SQS queue that triggers the function, and b) the reference to the source code.

The AWS Lambda functions ProcessTopK() and CalculateSimilarity() request information from DyanamoDB tables and implement a cache system in order to reduce the number of requests to DynamoDB. This is useful, when the message received from SQS contains more than one element

[1] https://www.pcap4j.org/

TABLE 9.1. SETUP OF AWS SERVICES EMPLOYED

Service	**Config**
SQS Queue Type	Standard
DynanoDB Capacity	OnDemand
SQS-to-Lambda Batch Size	10 elements
Lambda Runtime	Python 3.6
Lambda Memory Limit(all functions)	128 MB
Lambda Memory Limit(CalculateSimilarity function)	512 MB
Lambda Timeout	15 minutes

TABLE 9.2. S3 BUCKET NOTIFICATIONS DESTINATION

Bucket Name	**Notification Destination**
TopK Files	SQS: TopK Filenames
Split Filenames	SQS: Split Filenames

from the same user. For example, in function CalculateSimilarity(), the profile of a user is requested to DynamoDB only once. This has a positive impact in the running time of the function.

9.6. Experiments and Results

The proposed architecture was deployed into AWS. The *Capture Application* was installed in six different computers. This application was configured to automatically upload the TopK files every time it is executed; after that, it starts capturing packets. A three-week period was set as the

TABLE 9.3. AWS LAMBDA FUNCTIONS

Function	**Triggered event**	**Code**
SplitFiles()	SQS: TopK Filenames	Algorithm: 7
ReadTopKs()	SQS: Split Filenames	Algorithm: 8
ProcessTopK()	SQS: TopKs to process	Algorithm: 9
CalculateSimilarity()	SQS: TopKs to evaluate	Algorithm: 10

Algorithm 7 SplitFile.py

```
import json
import boto3

bucketNameDst = 'Split-Filenames'
s3 = boto3.resource('s3')
s3client = boto3.client('s3')

def lambda_handler(event, context):
    for r in event["Records"]:
        msg = json.loads(r["body"])
        if("Event" in msg and msg["Event"]=="s3:TestEvent"):
            continue
        for e in msg["Records"]:
            bucket = e["s3"]["bucket"]["name"]
            fileName = e["s3"]["object"]["key"]

            file = s3.Object(bucket,fileName)
            ruta,filename = file.key.split('/')
            base,ext = filename.split('.')

            oLines = file.get()['Body'].read().decode('utf-8').splitlines()
            lines_per_file=300
            created_files=0
            sfilelines=''
            for rownum,line in enumerate(oLines,start=1):
                sfilelines = sfilelines + line + '\n'
                if rownum%lines_per_file == 0:
                    cnt = created_files + 1
                    body_contents = str(sfilelines)
                    target_file = "%s/%s_%s_part.dat" % (ruta,base,cnt)
                    s3client.put_object(Bucket=bucketNameDst,
                                Key=target_file, Body=body_contents)
                    sfilelines = ''
                    created_files += 1

            if rownum:
                cnt = created_files + 1
                body_contents = str(sfilelines)
                target_file = "%s/%s_%s_part.dat" % (ruta,base,cnt)
                s3client.put_object(Bucket=bucketNameDst, Key=target_file,
                                                    Body=body_contents)
                created_files += 1

    return {
        'statusCode': 200,
        'body': json.dumps("Finish spliting files")
    }
```

Algorithm 8 ReadTopKs.py

```
import json
import boto3

s3 = boto3.resource('s3')
s3client = boto3.client('s3')
sqs = boto3.resource('sqs')

queue = sqs.get_queue_by_name(QueueName='TopKs-to-Process')

def lambda_handler(event, context):
    for r in event["Records"]:
        msg = json.loads(r["body"])
        for e in msg["Records"]:
            bucket = e["s3"]["bucket"]["name"]
            fileName = e["s3"]["object"]["key"]

            file = s3.Object(bucket,fileName)
            user,filename = file.key.split('/')
            base,ext = filename.split('.')

            oLines = file.get()['Body'].read().decode('utf-8').splitlines()
            i = 0
            for line in oLines:
                tokens = line.split("\t")
                timestamp = int(tokens.pop(0))

                msg = {
                    "user": user,
                    "timestamp": timestamp,
                    "topk": " ".join(tokens)
                }

                response = queue.send_message(MessageBody=json.dumps(msg))
                i+=1
    return {
        'statusCode': 200,
        'body': json.dumps("Finish spliting files")
    }
```

Algorithm 9 ProcessTopK.py

```
import json
import boto3
from boto3.dynamodb.conditions import Key, Attr

ranges = []

dynamo = boto3.resource('dynamodb')
tableProfile = dynamo.Table('phd_profile')
tableRange = dynamo.Table('phd_profile_ranges')

def noRangeDef_fail(event, context):
    raise Exception('No profile range defined for user')

def getProfileRange(user):
    for r in ranges:
        if r['user'] == user:
            return r['min'], r['max']

    res = tableRange.query(KeyConditionExpression=Key('user').eq(user))
    if(res['Count'] < 1):
        noRangeDef_fail(event, context)

    max = res['Items'][0]['max']
    min = res['Items'][0]['min']
    ranges.append({"user": user, "min": min, "max": max})
    return min, max

def lambda_handler(event, context):
    sqs = boto3.resource('sqs')
    queue = sqs.get_queue_by_name(QueueName='phd_topk_AO')

    for r in event["Records"]:
        msg = json.loads(r["body"])
        timestamp = msg["timestamp"]
        min, max = getProfileRange(msg["user"])

        if(msg["timestamp"] >= min and msg["timestamp"] < max):
            tableProfile.put_item(
                Item = {
                    'user': msg["user"],
                    'topk': msg["topk"]
                    }
            )
        else:
            response = queue.send_message(MessageBody=json.dumps(msg))

    return {
        "statusCode": 200,
        "body": json.dumps('Registro Insertado')
    }
```

Algorithm 10 CalculateSimilarity.py

```
import json
import boto3
from decimal import Decimal
from boto3.dynamodb.conditions import Key, Attr

dynamo = boto3.resource('dynamodb')
tableProfile = dynamo.Table('profile_table')
tableAO = dynamo.Table('similarity')
topkIN = []
cacheProfile = []

def getProfile(user):
    for p in cacheProfile:
        if p['user'] == user:
            return p['profile']
    profile = {"user": user, "profile": []}
    res = tableProfile.query(KeyConditionExpression=Key('user').eq(user))
    for item in res['Items']:
        profile['profile'].append(item['topk'])
    while 'LastEvaluatedKey' in res:
        res = tableProfile.query(KeyConditionExpression=Key('user').eq(user),
                                 ExclusiveStartKey=res['LastEvaluatedKey'])
        for item in res['Items']:
            profile['profile'].append(item['topk'])
    cacheProfile.append(profile)
    return profile["profile"]

def processItems(topkIN, items):
    maxAO = 0
    for i in items:
        ao = calcAO(topkIN, i.split())
        maxAO = ao if ao > maxAO else maxAO
    return maxAO

def calcAO(topkProfile, topk):
    maxDeep = len(topkProfile) if (len(topkProfile) > len(topk)) else len(topk)
    sumA = 0
    for d in range(1,(maxDeep+1)):
        sumA += calculateA(topkProfile[0:d],topk[0:d],d)
    return sumA/maxDeep

def calculateA(topkProfile, topk, deep):
    i = len(list(set(topkProfile) & set(topk)))
    return i/deep

def lambda_handler(event, context):
    for r in event["Records"]:
        msg = json.loads(r["body"])
        topkIN = msg["topk"].split()
        pro = getProfile(msg["user"])
        maxAO = processItems(topkIN, pro)
        msg["ao"] = Decimal(str(maxAO))
        tableAO.put_item(Item = msg)
    return { 'statusCode': 200, 'body': json.dumps('Hello from Lambda!') }
```

profile-building time. All the traffic occurred after that period of time was evaluated.

The *Capture Application* produces a total amount of 2880 TopKs during an eight-hour labor day. It takes around 190 seconds to process a file containing these TopKs. The approximate cost of TopK splitting, retrieval and evaluation is: $0.00869 USD. The breakdown of this cost is shown in Table 9.4.

TABLE 9.4. ONE LABOR-DAY COST BREAKDOWN USING SERVERLESS

Element	**Quantity**	**Cost(USD)**
Lambda: Request	799	$0.0001598
Lambda: Duration	7.05 GB/s	$0.0001176
DynamoDB: Read Request	492	$0.0001975
DynamoDB: Write Request	2880	$0.0036000
SQS Requests	11538	$0.0046152
Total		**$0.0086900**
Total execution time		**3.16 min**

With the aim of contrasting the performance of the architecture proposed, the proposed methodology was implemented using a different execution environment: an Amazon EC2 T3-micro instance, i.e., a virtual machine with 2 vCPU and 1 GB of RAM. The main differences between this implementation and the architecture presented before are the following: 1) it does not split the files, so the Split-File S3 bucket is not used, 2) no SQS queues were utilized, and 3) the TopK-Files bucket notifies the presence of a new file by means of an AWS SNS message. A file containing 2880 TopKs was processed using this environment. It took thirty six minutes to finish. The cost to process the file was 0.01054 USD. The breakdown of this cost is shown in Table 9.5.

9.7. Conclusion

In this chapter, we have proposed and implemented an AWS-based serverless execution architecture for the methodology first introduced in [60] that builds user network profiles and evaluates local network traffic. The AWS services proposed to be included are: S3, Lambda, DynamoDB and SQS.

In this implementation, all the components are 'as a Service', resulting in the following early

TABLE 9.5. ONE LABOR-DAY COST BREAKDOWN USING AMAZON EC2 INSTANCE

Element	**Quantity**	**Cost(USD)**
EC2 T3 Micro Instance	36 min	$0.0062718
EBS Storage	8 GB	$0.0006701
DynamoDB: Read Request	2	$0.0000005
DynamoDB: Write Request	2880	$0.0036000
Total		**$0.0105424**
Total execution time		**36.16 min**

benefits:

a) The capability to process all the information at any time with an availability of 99.99999%, without specifying any extra parameter or configuration.

b) Cost reduction by paying only for the amount of data processed. The idle time is not charged, as it happens when using virtual machines.

c) Time reduction, achieved by the implicit parallelism of lambda instances running independently in a serverless architecture.

d) Scalability. As the workload increases, the overall performance continues to function gracefully without further modifications. A serverless implementation applies a cost-effective strategy for extending hardware capacity.

10. Analysis of TopK Network Profile similarities for identification of Predominant Network Behavior

In accordance with the NIST, an that uses anomaly-based detection has profiles that represents the normal behavior of any of the following: user, host, network connection or application; then it is compared to real-time activity in order to detect a significant difference [68].

In [60], it is proposed a profiling method based on building TopK rankings of accessed services from network traffic captured at the host. Each service is represented by the 3-tuple <remote IP address, transport protocol, remote port>. This profiling process is carried out within a secure environment where we can guarantee that the host will be used only by the expected user and there are no malware, virus, trojan or any other malicious software installed. This method produces a profile structure constituted by a list of TopKs denoting the normal behavior of a user at their computer.

Each TopK in the profile represents the top K most accessed services based on total transferred bytes, during a timeframe f. A new TopK is calculated every Δt seconds. Each TopK is overlapped with the previous ones as explained in the first chapters.

Additionally, this profiling method offers a mechanism to determine how similar a given TopK ranking is to the profile, returning a value in the range $[0.0 \cdots 1.0]$, where 0.0 and 1.0 denotes, respectively, totally different and identical.

This chapter proposes a methodology capable to detect an unexpected network behavior -which might be an intrusion- based on computing the predominant behavior of the user.

10.1. Unexpected behavior identification

The proposed methodology capable to detect an unexpected network behavior based on computing the predominant behavior of the user is depicted in Figure 10.1 and consists of the following phases:

a) Capture continuously real-time network traffic at the host

b) Build a TopK ranking every Δt seconds from the most recently captured traffic.

c) Calculate similarity S of each TopK to the user profile.

d) Identify the predominant behavior every Δw seconds.

e) Evaluate the current predominant behavior.

f) Determine whether to trigger or not an alarm.

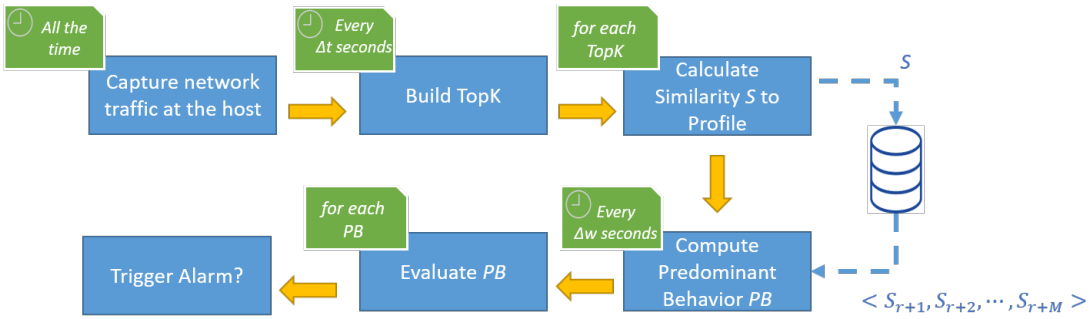


Figure 10.1: Block diagram of the proposed methodology for detecting an unexpected behavior

The first two phases employ the same algorithms and parameters as those used to build the user profile. The similarity is calculated using the mechanism offered by the profiling system[60], which is based on the Average Overlap measure[79].

Figure 10.2 depicts a sequence of S calculated during six hours capturing real-time traffic of a single user. We can observe that the points are too disperse to conclude that there is an unexpected behavior by evaluating a single similarity value S. Therefore, a method that analyzes many successive points will be useful to conclude whether the predominant behavior is actually unexpected or not.

In order to identify the predominant network behavior during a specific time-frame, we use a signal-processing technique called Moving-Average Filter that takes M points from x, that represents the points of the raw signal. Each point $x[l]$, where l is a time that belongs to time-frame(W) $[w1 \cdots w2]$. This filter reduces these points into a single point $y[n]$ calculating the mean value[52]. Formally:

$$y[n] = \frac{1}{M} \sum_{l=w_1}^{w_2} x[l] \tag{10.1}$$

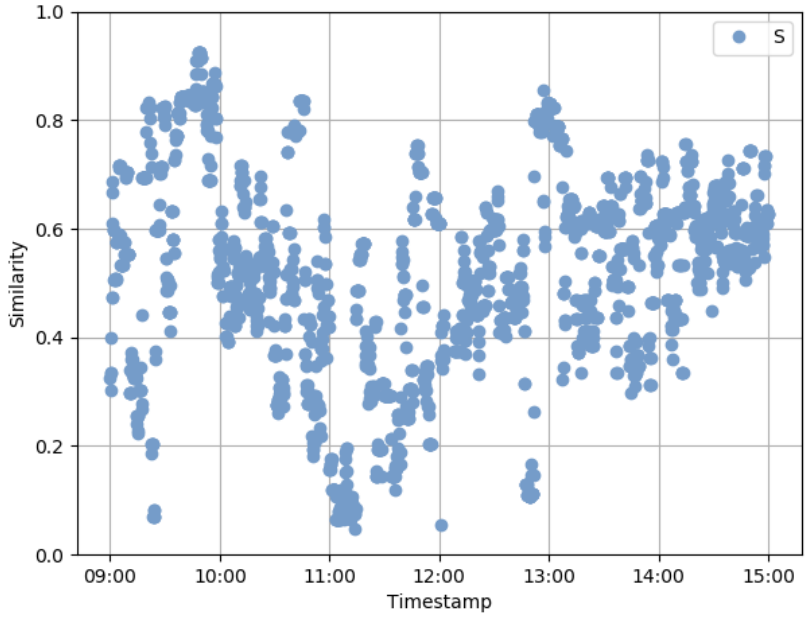


Figure 10.2: Sample sequence of similarity values obtained from 6-hour network traffic

The next time-frame starts at $w_1 + \Delta w$. In this implementation, Δw is smaller than $w_2 - w_1$ to guarantee that time-frames overlap in congruence with the methodology. Figure 10.3 depicts the operation of the Moving-Average Filter and how time frames overlap.

Figure 10.4 depicts the lines produced after connecting all the points obtained from applying the filter to the sequence of similarity values S (blue dots) shown in Figure 10.2. We can observe that each of these lines show the overall direction or trend of a sequence of similarity values. In the next section we explore different values of W and Δw, and different statistical functions.

10.2. Experiments

10.2.1 Search best parameters

The purpose of the first experiment is to find the parameters $\langle W, \Delta w, func \rangle$ such that the predominant behavior calculated at each time-frame best represents the expected behavior, where: W denotes the window width in minutes, Δw stands for the overlap length in seconds, and $func$ is the statistical function.

Six hours of real traffic was analyzed by the TopK profiling system which returned a set of similarity values S. The traffic analyzed includes 40 minutes of an identified unexpected behavior

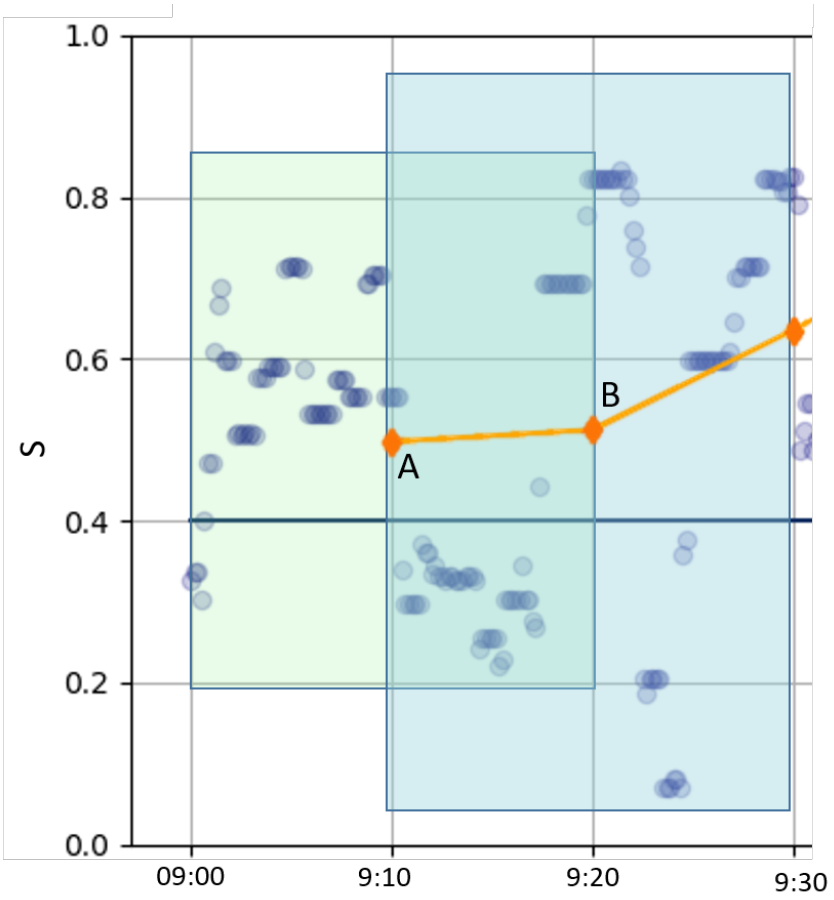


Figure 10.3: Moving-Average Filter. Diamond A represents the average of all the similarity values s from 9:00 to 9:20, and diamond B represents the average from 9:10 to 9:30

for which we know the start and end timestamps. Each similarity value $s \in S$ was classified as *during-expected* s_e or *during-attack* s_a based on its timestamp.

For each combination of parameters $\langle W, \Delta w, func \rangle$, we computed two sequences of predominant behaviors B_e, B_a from S. For each $b \in B_e$, the center of its window is out of the *attack* period of time. Conversely, for each $b \in B_a$, the center of its window is in this period.

The evaluation of the combination is in terms of how far each predominant behavior b is from the expected region, i.e., each $b \in B_e$ should be high enough to fit into the *expected-behavior* region, whereas a each $b \in B_a$ should be low enough to fit into the *under-attack* region.

We have defined a threshold value A_e that represents the minimum value for a b to fit in the *expected-behavior* region, and is calculated as the average of all the s_e values. For each $b \in B_e$ such that $b < A_e$, we calculate its distance to the *expected-behavior* region as $A_e - b$, and then we sum

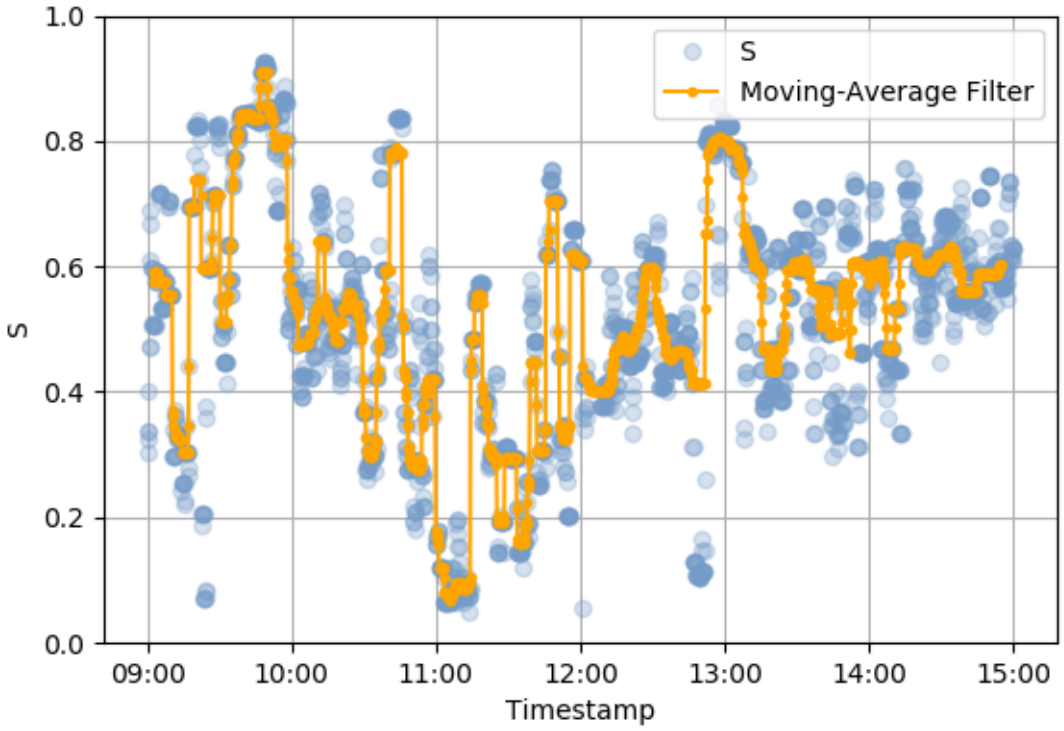


Figure 10.4: Applying Moving-Average Filter to a sequence of similarity values, with $W = 5m$ and $\Delta w = 10s$

up these distances to get a global distance to this region D_e (Equation 10.3). Notice that, if $b > A_e$, the distance of b to this region is zero.

Similarly, we have defined a threshold value A_a that represents the maximum value for a b to fit in the *under-attack* region, and is calculated as the average of all the s_a values. For each $b \in B_a$ such that $b > A_a$, we calculate its distance to the *under-attack* region as $b - A_a$, and sum up these distances to get a global distance to this region D_a (Equation 10.4).

Equation 10.2 calculates the average of all the distances D_e and D_a with respect to the total number of predominant behaviors.

Figure 10.5 depicts an example of a predominant-behavior calculation using a set of similarity values S that represents forty minutes of network traffic which includes a five-minute period of an attack. The predominant behaviors were obtained using the combination $< W = 60s, \Delta w =$

$10s, func = average >$. The horizontal lines represent the threshold values A_e and A_a. Every red point denotes either a predominant behavior $b \in B_e$ lying under threshold line A_e, or a predominant behavior $b \in B_a$ lying over the threshold line A_a.

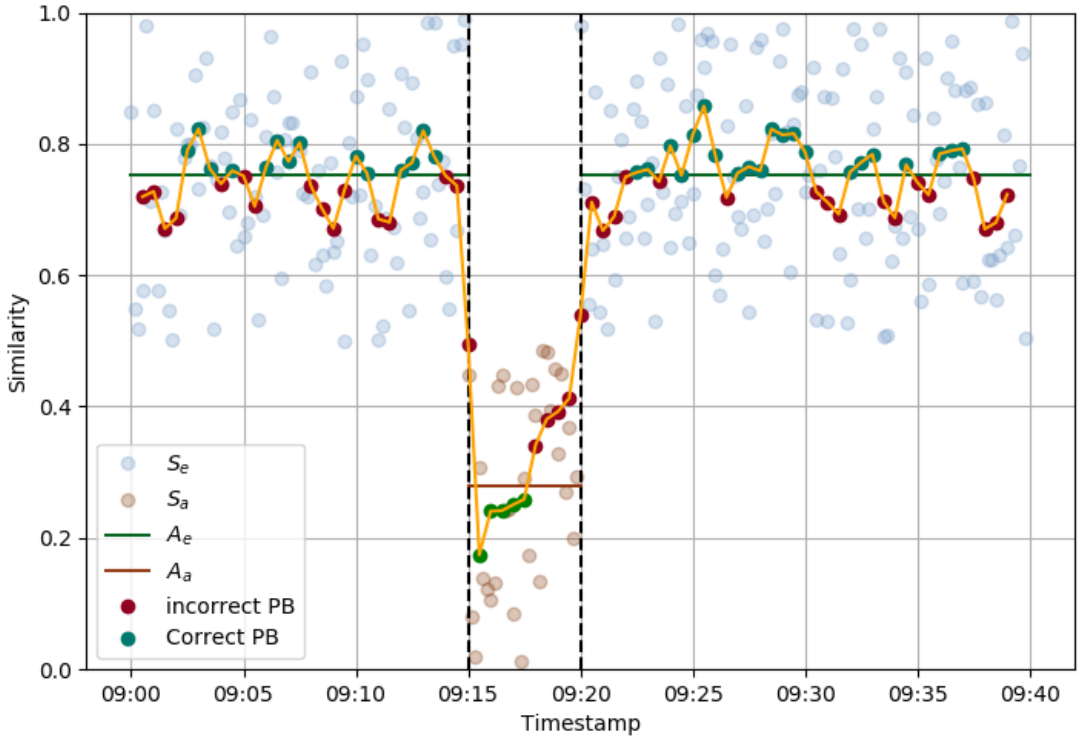


Figure 10.5: Calculating predominant behavior from network traffic that includes an attack. Parameters: <$W = 60s$, $\Delta w = 10s$, $func = avg >$

$$\mathbb{D}_{B_e,B_a} = \frac{D_e + D_a}{|B_e| + |B_a|} \tag{10.2}$$

$$D_e() = \sum_{b \in B_e} \begin{cases} 0, & \text{if} b > A_e \\ A_e - b, & \text{if} b <= A_e \end{cases} \tag{10.3}$$

$$D_a() = \sum_{b \in B_a} \begin{cases} 0, & \text{if} b < A_a \\ b - A_a, & \text{if} b >= A_a \end{cases} \tag{10.4}$$

We calculated $\mathbb{D}_{B_e,B_a}$ for each parameter combination $< W, \Delta w, func >$, where: $W \in \{5, 10, \cdots, 40, 45\}$, $\Delta w \in \{10, 20, 30, 60\}$, and $func \in \{average, median\}$. These combinations were ranked in non-ascending order of $\mathbb{D}_{B_e,B_a}$, such that the best combination is the one that

minimizes $\mathbb{D}_{B_e,B_a}$. Table 10.1 shows the calculated distances corresponding to the best, median and worst combinations. Figure 10.6 shows the sequence of predominant behaviors produced from each of these combinations.

TABLE 10.1. CALCULATING $\mathbb{D}$ FOR THREE SETS OF PARAMETERS

Rank	W	ΔW	$func$	D_a	D_e	$\lvert B_e + B_a \rvert$	$\mathbb{D}$
1^{st} *of* 72	20m	10s	avg	65.57	8.88	2007	0.0371
36^{th} *of* 72	30m	10s	med	73.09	7.096	1947	0.0411
72^{th} *of* 72	5m	60s	med	19.29	2.38	355	0.0610

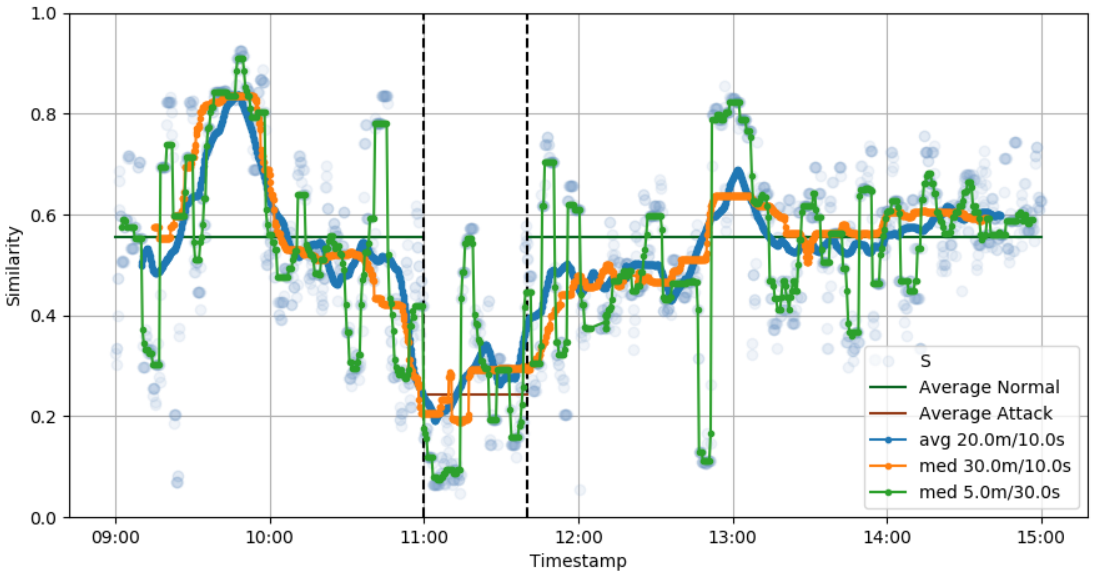


Figure 10.6: Calculating predominant Behavior calculated with three different set of parameters

10.2.2 Methodology Validations

The profile of four different users A, B, C, and D was built using the TopK profiling technique. Next, real-time traffic of each user was analysed; then, the predominant behavior was computed. With the purpose of generating an unexpected behavior, a malware was installed on each computer.

The malware was created with the Metaexploit[1] framework using a reverse HTTPS Meterpreter payload that connects to the Metaexploit server, which was hosted outside the LAN. The malware

[1]https://www.metasploit.com/(visited: Mar 14, 2019)

transferred files from the user computer to an external server. After copying one GB of data, the malware finished its execution and removed itself.

The real-time traffic of each user was analyzed independently by the TopK profiling system which returned a set of similarity values S. Each similarity value $s \in S$ was classified as *during-expected* s_e or *during-attack*, s_a based on its timestamp and the period of time of the malware.

For each user, we computed two sequences of predominant behaviors B_e, B_a from S. For each $b \in B_e$, the center of its window is out of the *malware-execution* period of time. Conversely, for each $b \in B_a$, the center of its window is in this period.

In order to evaluate the capability of the methodology to identify attacks, we implemented the same technique used first experiment, with the founded parameters.

Figures 10.7a-10.7d depicts the predominant network behavior of users a-d during the execution of the malware. The start and end times of the malware execution is denoted by the vertical black lines. Table 10.2 shows the $\mathbb{D}$ of each user.

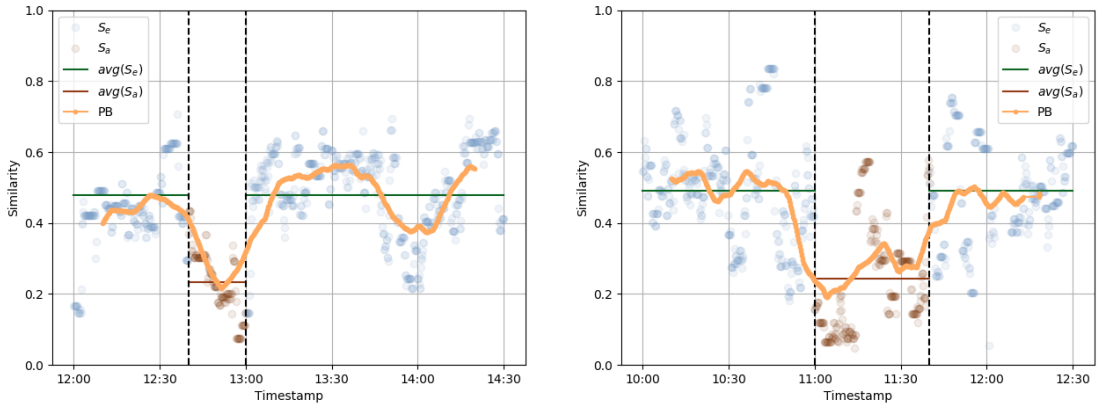
TABLE 10.2. AVERAGE DISTANCE OF WRONG POINTS TO EXPECTED AREA

User	$\mathbb{D}$
A	0.0345
B	0.0333
C	0.0198
D	0.0195

10.3. Conclusion

In this chapter, we have presented the results of applying a discrete signals filter technique to analyze the sequence of similarity values S. The Moving-Average filter summarizes a block of S into a single point that represents the predominant behavior during a time frame.

The implemented filter admits three parameters W,Δw and a statistical function. We applied this filter using different values for each parameter. Then, we evaluated each of the results using a formula that calculates the average error of all the incorrect points. The best results were produced with the following parameters: $W = 20m$, $\Delta w = 10s$ and $func = average$.



(a) User A

(b) User B

(c) User C

(d) User D

Figure 10.7: Predominant behavior of five different users during a malware attack

Then we tested the capability of using Moving-Average filter to identify the predominant network behavior from a sequence of S in terms of how far each predominant behavior b is from the expected region.

As can observed in table 10.2, we obtained similar results after averaging the distances of each point b to its expected region $\mathbb{D}$ on four different users. Thus, the parameters selected to calculate and evaluate the predominant behavior proved to be consistent across different users.

From Figures 10.7a-10.7d, we can observe that the curve generated after connecting all bs has a valley during the attack, as it is expected.

11. Unexpected behavior detection using TopK Rankings

In Chapter 10, it is proposed a methodology capable to detect an unexpected network behavior -which might be an intrusion- based on computing the predominant behavior of the user, by means of *moving-average* filter.

The methodology compares real-time traffic of the user with his/her profile and calculates the predominant behavior b, which represents twenty minutes of traffic. A new predominant behavior is calculated and registered every ten seconds.

To determine whether the owner of the traffic is having or not an expected behavior, it is necessary to classify each calculated b as either *positive* or *negative*. This requires a threshold value T that represents the minimum value of b to be classified as *positive*.

In this chapter, we evaluate the precision of different threshold values, which are applied over four sequences of predominant behaviors previously classified.

11.1. Evaluation of Predominant Behavior Classification

To find the best value of T, we require real-time traffic of different users, such that each traffic includes an identified period of time during which an unexpected behavior occurred (i.e., an attack). Thus, each calculated b can be classified in advance as *positive* or *negative*, based on the timestamps.

Prior to evaluate each T, each b is assigned one of the following tags: 1) True-positive, if it is previously classified as *positive* and satisfies: $b >= T$, 2) True-negative, if it is classified as *negative* and satisfies: $b < T$, 3) False-positive(Error Type I), if it is classified as *negative* and does not satisfy: $b < T$, 2) False-negative(Error Type II), if it is classified as *positive* and does not satisfy: $b >= T$. Table 11.1 summarizes each of these tags, and Fig 11.1 illustrates the region where each one belongs.

A natural characteristic of a sequence of predominant behaviors is that most of them represent expected behaviors, whereas a small proportion denote unexpected ones. Since we are dealing with an unbalanced dataset, we cannot apply *Accuracy* measure to calculate the precision of the classification.

TABLE 11.1. TAGGING PREDOMINANT BEHAVIORS

		Threshold-based classification	
		Expected Behavior	Under Attack
Prev. Class.	Expected Behavior	True-Positive	False-Negative(Type II)
	Under attack	False-Positive(Type I)	True-Negative

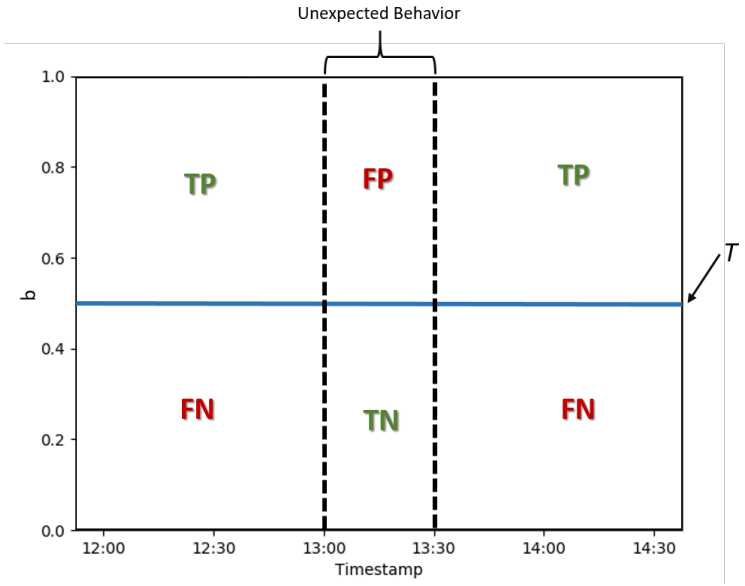


Figure 11.1: Classification of each b based on T and the timestamp

Some statistical measures to evaluate the performance of binary classifiers for unbalanced datasets include the following: 1) Sensitivity, also known as true-positive rate (TPR), which measures the proportion of positives correctly classified; and 2) Specificity, also known as true-negative rate (TNR), which measures the proportion of negatives correctly identified. Equations 11.1 and 11.2 expresses each of these measures, where TP, TN, FP and FN stand for the count of predominant behaviors tagged as true-positive, true-negative, false-positive and false-negative, respectively.

$$Sensitivity = \frac{TP}{TP + FN} \tag{11.1}$$

$$Specificity = \frac{TN}{TN + FP} \tag{11.2}$$

Two ways to combine these measures for getting a unique score on the performance of a classification are the following: $BalancedAccuracy$ (Eq. 11.3) and the $G - Mean$ (Eq. 11.4).

$$BalancedAccuracy = \frac{Sensitivity + Specificity}{2} \tag{11.3}$$

$$G - Mean = \sqrt{Sensitivity * Specificity} \tag{11.4}$$

In this work, we apply both measures and compare the results from each.

11.2. Experiments and Results

First, the profile of four different users A, B, C, and D was built using the TopK profiling technique (See Chapter 9. Next, real-time traffic of each user was processed; then, the sequence of predominant behaviors was computed during a labor week. With the purpose of generating an unexpected behavior, a malware was installed on each computer during a controlled period of time.

The malware was created with the Metaexploit[1] framework using a reverse HTTPS Meterpreter payload that connects to the Metaexploit server, which was hosted outside the LAN. The malware transferred files from the user computer to an external server. After copying one GB of data, the malware finished its execution and removed itself.

The remaining problem consists in finding the threshold T in the range $[0.00 \cdots 1.00]$ that maximizes the classification score for each user.

For every user, we calculated *Balanced Accuracy* and *G-mean* using each $T = 0.00, 0.01, \cdots, 0.99, 1.00$. We recorded the values of $Sensitivity$, $Specificity$ and the threshold T^* that maximized both scores. These values can be appreciated in Table 11.2. From the table, we can notice the following: a) both scores are maximized by the same T^* on a particular user, b) a different T^* was found for each user, and c) all the values of T^* were found in the interval [0.3...0.4].

With the aim of finding a single threshold that works fine for all the users, we averaged the scores *Balanced Accuracy* and *G-mean* of the four users for each threshold T in the range $[0.00 \cdots 1.00]$. We found that the threshold with the highest average for both scores is 0.37. Fig. 11.2 plots this average for each T, and also includes the averages of $Sensitivity$ and $Specificity$.

[1] https://www.metasploit.com/

TABLE 11.2. MAXIMUM SCORES FOR EACH USER.

	Sensitivity			**Specificity**			**BA**		**G-Mean**	
User	TP	FN	TPR	TN	FP	TNR	T^*	Score	T^*	Score
A	6955	2450	73.95%	102	19	84.30%	0.35	0.7912	0.35	0.7895
B	18636	213	98.87%	241	0	100%	0.39	0.9944	0.39	0.9943
C	8391	15	99.83%	332	0	100%	0.31	0.9991	0.31	0.9991
D	5959	866	87.31%	199	41	82.92%	0.39	0.8511	0.39	0.8509

For each user, Table 11.3 contrasts the *G-mean* score obtained using the best individual threshold with that obtained using the best global threshold (0.37). We can see that *G-mean* decreases by around 0.01 in all the cases.

TABLE 11.3. COMPARING BEST LOCAL AND GLOBAL T FOR EACH USER

User	T	$Gmean$
A	0.35	0.7895
	0.37	0.7768
B	0.39	0.9943
	0.37	0.9871
C	0.31	0.9992
	0.37	0.9877
D	0.39	0.8509
	0.37	0.8416

11.3. Conclusions

In this chapter, we presented the evaluation of the TopK Profiling method with predominant behavior calculation for binary classification of network traffic. This evaluation has been done by calculating True Positive Rate, True Negative Rate and G-mean score.

Every predominant behavior b is classified as either $positive$ or $negative$ according to a threshold T that represents the minimum value for any b to be considered as an expected behavior, i.e., $positive$.

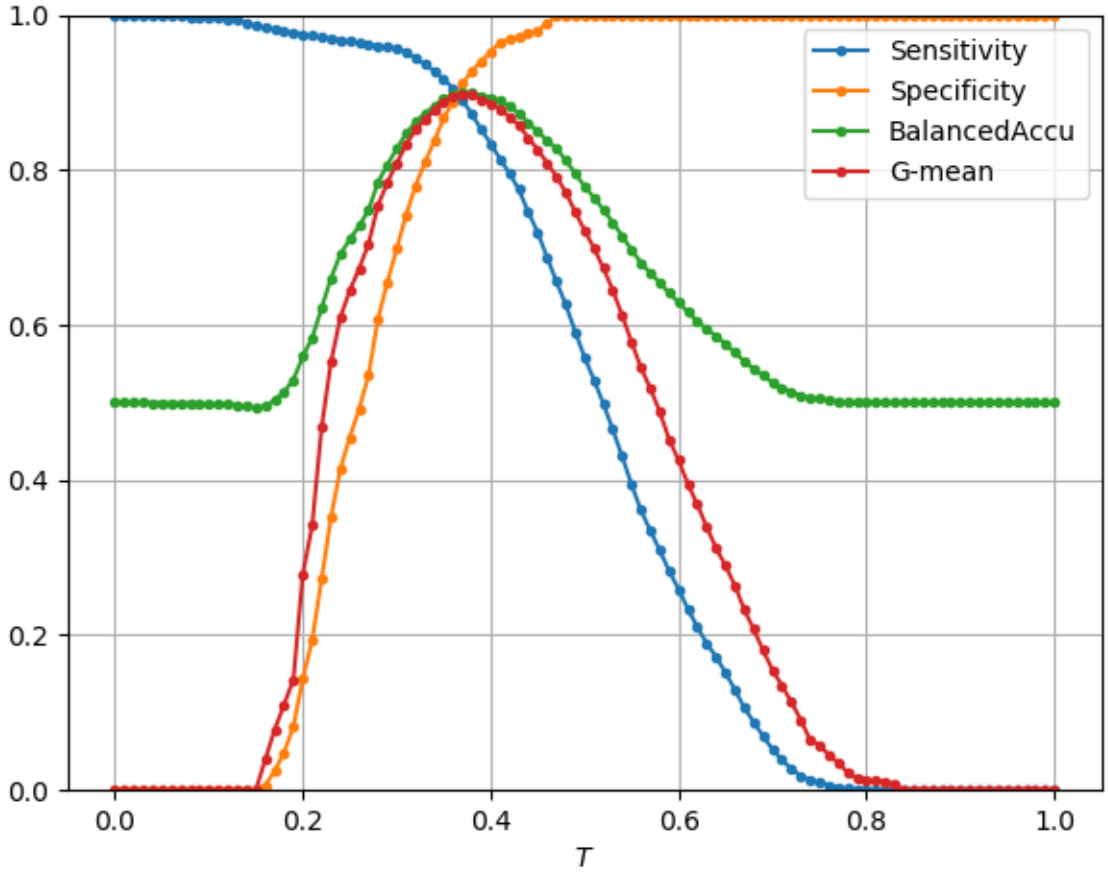


Figure 11.2: Average $Sensitivity$, $Specificity$, $BalancedAccuracy$ and $G-Mean$ of the four datasets

We have calculated the threshold that maximizes the G-Mean score for each user. These thresholds range from 0.31 to 0.39. In addition, we found a global threshold $T = 0.37$ that maximizes the average G-Mean. The individual G-Mean scores obtained with the global threshold are very similar to those obtained with the best threshold per user, i.e., the difference between each is less than 0.012.

It is important to notice that, in some cases $TNR = 100\%$, and it is never less than 80%. This leads us to think that if we implement this method on an we will expect a good performance on detecting intrusions.

General Conclusions

This doctoral dissertation presents a user-level anomaly-based intrusion-detection methodology that processes network traffic exclusively at the user's host with the aim of identifying unexpected behaviors that might be attacks.

In a CAN, like the networks at universities, there exist a number of users who have certain privileges on information systems that are critical to the operation of the entire organization. It is known that traditional layers of security, such as firewalls, antivirus, , among others, could be violated by either unidentified or rarely-known attacks. Therefore, it is important to protect privileged users by adding an additional layer of security, like an anomaly-based intrusion detection system.

From the review of a number of state-of-the-art research works on anomaly-based intrusion detection systems, it was clearly noticeable that there are quite a lot of works around this topic. However, as a result of a deeper analysis of these works, we realized that most of them are focused on testing Machine Learning (ML) algorithms, instead of on detecting new types of attacks. A common feature of ML-centered works is that they all use synthetic network traffic data sets; in contrast, the works focused on detecting new types of attacks mostly employ data sets that are built from real traffic.

The proposed methodology differs from the state of the art in the following: 1) it works exclusively with the host's network traffic instead of the entire network segment, and 2) it does not process system calls or any other parameter other than traffic network to perform anomaly detection.

Based on the experimental results, we can conclude the following:

a) A trojan malware execution affects the network behavior at a given host, causing a significant reduction of the similarity value between real-time traffic and the profile. Therefore, our methodology is capable of triggering an alarm when the predominant behavior of the user starts deviating from the expected one.

b) An anomaly-based IDS that builds a profile for each individual network user represents an additional security mechanism because it is capable of detecting unexpected network behaviors

that might be originated by a malware. The antivirus was actually not capable to detect the installed Trojan in either of our experiments.

An anomaly-based IDS must update the profiles on a regular basis because the normal behaviors of users change periodically. The proposed profiling method relies on the creation of TopK lists instead of using a supervised classifier as other approaches do. Therefore, updating the profile is computationally viable because it does not involve a re-training process.

Future work can lead to the design of a dynamic profiling method capable of: 1) removing the least common behaviors from the profile, which might include the behavior induced by an attack that occurred during a previous profiling process, and 2) adding new network behaviors that might denote an update of the regular activity of the user.

Conclusiones Generales

En este trabajo de tesis doctoral se ha presentado una metodología de detección de intrusos basada en anomalías y a nivel de usuario. La metodología procesa el tráfico de red en el host, exclusivamente, con el objetivo de identificar comportamientos inesperados que podrían suponer ataques.

En una CAN, como las que tienen las universidades, existen usuarios con ciertos privilegios sobre los sistemas de información que son críticos para la operación de la organización. Es sabido que las capas tradicionales de seguridad como son firewalls, antivirus, IDS, entre otros, pueden ser vulneradas por ataques poco conocidos o no identificados. Por lo tanto, es importante proteger a estos usuarios privilegiados añadiendo una capa adicional de seguridad como puede ser un sistema de detección de intrusos basado en anomalías.

A partir de la revisión de diversos trabajos de investigación sobre sistemas de detección de intrusos basados en anomalías, se pudo observar que existen muchos trabajos alrededor de este tema. Mediante un estudio más profundo de estos trabajos, se pudo observar que la mayoría de ellos están enfocados en evaluar el desempeño de algoritmos de *Machine Learning*, en lugar de en detectar nuevos tipos de ataques. Una característica común de los trabajos centrados en algoritmos de ML es que utilizan conjuntos de datos de tráfico de red sintéticos, mientras que los trabajos enfocados en la detección de nuevos tipos de ataques suelen emplear conjuntos de datos provenientes de trafico real.

El trabajo propuesto se diferencia del estado del arte en lo siguiente: 1) trabaja exclusivamente con el tráfico de red del host, en lugar de trabajar con todo el segmento de red, y 2) no considera llamadas al sistema operativo ni algún otro parámetro diferente al tráfico de red para realizar la detección de anomalías.

De acuerdo con los resultados obtenidos de los experimentos realizados durante este trabajo, se puede concluir lo siguiente:

a) Un *Malware* de tipo troyano afecta el comportamiento de la red en un host determinado, lo que provoca una reducción significativa del valor de similitud entre el tráfico en tiempo real y su perfil. La metodología propuesta es capaz de activar una alarma cuando el comportamiento

predominante del usuario comienza a alejarse del comportamiento esperado.

b) Un IDS basado en anomalías que crea un perfil para cada usuario individual de la red representa un mecanismo de seguridad adicional porque es capaz de detectar comportamientos inesperados de la red que podrían ser originados por un malware. El antivirus no fue capaz de detectar el troyano instalado en ninguno de nuestros experimentos.

Un IDS basado en anomalías debe actualizar los perfiles de forma regular dado que los comportamientos normales de los usuarios cambian periódicamente. El método de creación de perfiles propuesto se basa en la creación de listas TopK en lugar de utilizar un clasificador supervisado como lo hacen otros enfoques. Por lo tanto, la actualización del perfil es computacionalmente viable porque no implica un proceso de reentrenamiento.

El trabajo futuro puede dirigirse hacia el diseño de un método de creación de perfiles dinámico capaz de: 1) eliminar los comportamientos menos comunes contenidos en el perfil, los cuales podrían incluir aquellos inducidos por algún ataque que ocurrió durante el proceso de perfilado, y 2) agregar nuevos comportamientos que representen una actualización de la actividad normal del usuario.

Appendix

A. List of Internal Research Reports

- A. I. Parres-Peredo, H. I. Piza-Davila, and F. Cervantes-Alvarez, "Challenges and opportunities in computer network security," Internal Report PhDEngScITESO-15-20-R, ITESO, Tlaquepaque, Mexico, Dec. 2015.

- A. I. Parres-Peredo, H. I. Piza-Davila, and F. Cervantes-Alvarez, "A novel user network profile based on host network traffic," Internal Report PhDEngScITESO-16-08-R, ITESO, Tlaquepaque, Mexico, Jul. 2016.

- A. I. Parres-Peredo, H. I. Piza-Davila, and F. Cervantes-Alvarez, "Profiling network traffic for internal security using top-k ranking similarity measures," Internal Report PhDEngScITESO-16-17-R, ITESO, Tlaquepaque, Mexico, Dec. 2016.

- A. I. Parres-Peredo, H. I. Piza-Davila, and F. Cervantes-Alvarez, "Comparing user network profiles by means of top-k ranking similarity measures," Internal Report PhDEngScITESO-17-13-R, ITESO, Tlaquepaque, Mexico, May 2017.

- A. I. Parres-Peredo, H. I. Piza-Davila, and F. Cervantes-Alvarez, "Algorithm to Calculate Overlapping Top-K Rankings from Network Traffic," Internal Report PhDEngScITESO-17-51-R, ITESO, Tlaquepaque, Mexico, Dec. 2017.

- A. I. Parres-Peredo, H. I. Piza-Davila, and F. Cervantes-Alvarez, "Map-reduce approach to build network user profiles with top-k rankings," Internal Report PhDEngScITESO-17-54-R, ITESO, Tlaquepaque, Mexico, Dec. 2017.

- A. I. Parres-Peredo, H. I. Piza-Davila, and F. Cervantes-Alvarez, "Implementation and validation of mapreduce algorithm to build Network user profiles with top-k rankings," Internal Report PhDEngScITESO-17-57-R, ITESO, Tlaquepaque, Mexico, Dec. 2017.

- A. I. Parres-Peredo, H. I. Piza-Davila, and F. Cervantes-Alvarez, "A serverless architecture for building and evaluating topk user network profiles for cybersecurity," Internal Report PhDEngScITESO-18-42-R, ITESO, Tlaquepaque, Mexico, Dec 2018.

- A. I. Parres-Peredo, H. I. Piza-Davila, and F. Cervantes-Alvarez, "Implementation and validation of serverless architecture for Building and evaluating topk user network profiles for Cybersecurity," Internal Report PhDEngScITESO-18-46-R, ITESO, Tlaquepaque, Mexico, Dec 2018.

- A. I. Parres-Peredo, H. I. Piza-Davila, and F. Cervantes-Alvarez, "Analysis of top-k network profile similarities for identification of predominant network behavior," Internal Report PhDEngScITESO-18-56-R, ITESO, Tlaquepaque, Mexico, Dec. 2018.

- A. I. Parres-Peredo, H. I. Piza-Davila, and F. Cervantes-Alvarez, "Validation of moving-average filter for identification of predominant network behavior," Internal Report PhDEngScITESO-19-25-R, ITESO, Tlaquepaque, Mexico, Dec. 2019.

- A. I. Parres-Peredo, H. I. Piza-Davila, and F. Cervantes-Alvarez, "Unexpected behavior detection using topk rankings," Internal Report PhDEngScITESO-19-29-R, ITESO, Tlaquepaque, Mexico, Dec. 2019.

- A. I. Parres-Peredo, H. I. Piza-Davila, and F. Cervantes-Alvarez, "Review of state of the art of anomalies based intrusion detection systems," Internal Report PhDEngScITESO-19-33-R, ITESO, Tlaquepaque, Mexico, Dec. 2019.

B. List of Publications

- A. Parres-Peredo, I. Piza-Davila and F. Cervantes, ”Towards a user network profiling for internal security using top-k rankings similarity measures,” 2017 40th International Conference on Telecommunications and Signal Processing (TSP), Barcelona, 2017, pp. 16-19. doi: 10.1109/TSP.2017.807592

- A. Parres-Peredo, I. Piza-Davila and F. Cervantes, ”MapReduce Approach to Build Network User Profiles with Top-k Rankings for Network Security,” 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, 2018, pp. 1-5. doi: 10.1109/NTMS.2018.8328702

- A. Parres-Peredo, I. Piza-Davila and F. Cervantes, ”Building and Evaluating User Network Profiles for Cybersecurity Using Serverless Architecture,” 2019 42nd International Conference on Telecommunications and Signal Processing (TSP), Budapest, Hungary, 2019, pp. 164-167. doi: 10.1109/TSP.2019.8768825

- A. Parres-Peredo, I. Piza-Davila, and F. Cervantes, “Unexpected-Behavior Detection Using TopK Rankings for Cybersecurity,” Applied Sciences, vol. 9, no. 20, p. 4381, Oct. 2019.

Appendix B. List of Publications

Bibliography

[1] *2015 Annual Security Report*. English. Annual Security Report. San Jose, Ca.: CISCO, 2015.

[2] O. Abdel Wahab et al. "Resource-Aware Detection and Defense System Against Multi-Type Attacks in the Cloud: Repeated Bayesian Stackelberg Game". In: *IEEE Transactions on Dependable and Secure Computing* (2019), pp. 1–1. ISSN: 2160-9209. DOI: 10.1109/TDSC.2019.2907946.

[3] R. Abdulhammed et al. "Deep and Machine Learning Approaches for Anomaly-Based Intrusion Detection of Imbalanced Network Traffic". In: *IEEE Sensors Letters* 3.1 (Jan. 2019), pp. 1–4. ISSN: 2475-1472. DOI: 10.1109/LSENS.2018.2879990.

[4] *About AWS*. https://aws.amazon.com/about-aws/. [Online; accessed Jan-20-2018].

[5] Ajay Kumara M.A and Jaidhar C.D. "Hypervisor and virtual machine dependent Intrusion Detection and Prevention System for virtualized cloud environment". In: *2015 1st International Conference on Telematics and Future Generation Networks (TAFGEN)*. May 2015, pp. 28–33. DOI: 10.1109/TAFGEN.2015.7289570.

[6] W. Alhakami et al. "Network Anomaly Intrusion Detection Using a Nonparametric Bayesian Approach and Feature Selection". In: *IEEE Access* 7 (2019), pp. 52181–52190. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2912115.

[7] S. A. Althubiti, E. M. Jones, and K. Roy. "LSTM for Anomaly-Based Network Intrusion Detection". In: *2018 28th International Telecommunication Networks and Applications Conference (ITNAC)*. Nov. 2018, pp. 1–3. DOI: 10.1109/ATNAC.2018.8615300.

[8] *Amazon API Gateway*. https://aws.amazon.com/api-gateway. [Online; accessed Jan-20-2018].

[9] *Amazon DynamoDB*. https://aws.amazon.com/dynamodb/. [Online; accessed Jan-20-2018].

[10] *Amazon S3*. https://aws.amazon.com/s3/. [Online; accessed Jan-20-2018].

[11] *Amazon Simple Queue Service*. https://aws.amazon.com/sqs/. [Online; accessed Jan-20-2018].

[12] M. Anandapriya and B. Lakshmanan. "Anomaly Based Host Intrusion Detection System using semantic based system call patterns". In: *2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO)*. Jan. 2015, pp. 1–4. DOI: 10.1109/ISCO.2015.7282244.

[13] James P. Anderson. *Computer Security Threat Monitoring And Surveillance*. Report Contract 79F296400. Fort Washington, PA: NIST, 1980. URL: http://csrc.nist.gov/publications/history/ande80.pdf.

[14] K. Ariyapala et al. "A Host and Network Based Intrusion Detection for Android Smartphones". In: *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. Mar. 2016, pp. 849–854. DOI: 10.1109/WAINA.2016.35.

[15] *AWS Lambda*. https://aws.amazon.com/lambda. [Online; accessed Jan-20-2018].

[16] Adrian Badea, Victor Croitoru, and Daniel Gheorghica. "Computer networks security based on the detection of user's behavior". In: *9th International Symposium on Advanced Topics in Electrical Engineering (ATEE)*. Bucharest, Romania: IEEE, May 2015, pp. 55–60.

[17] S. Bijani and M. Kazemitabar A. "HIDMN: A Host and Network-Based Intrusion Detection for Mobile Networks". In: *2008 International Conference on Computer and Electrical Engineering*. Dec. 2008, pp. 204–208. DOI: 10.1109/ICCEE.2008.183.

[18] C. Callegari et al. "Real Time Attack Detection with Deep Learning". In: *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. June 2019, pp. 1–5. DOI: 10.1109/SAHCN.2019.8824811.

[19] Y. Chae, N. Katenka, and L. DiPippo. "An Adaptive Threshold Method for Anomaly-based Intrusion Detection Systems". In: *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*. Sept. 2019, pp. 1–4. DOI: 10.1109/NCA.2019.8935045.

[20] G. Creech and J. Hu. "A Semantic Approach to Host-Based Intrusion Detection Systems Using Contiguousand Discontiguous System Call Patterns". In: *IEEE Transactions on Computers* 63.4 (Apr. 2014), pp. 807–819. ISSN: 2326-3814. DOI: 10.1109/TC.2013.13.

[21] Canadian Institute for Cybersecurity. *CSE-CIC-IDS2018*. 2018. URL: https://www.unb.ca/cic/datasets/ids-2018.html (visited on 10/01/2019).

[22] Canadian Institute for Cybersecurity. *Intrusion Detection Evaluation Dataset*. 2017. URL: https://www.unb.ca/cic/datasets/ids-2017.html (visited on 10/01/2019).

[23] Canadian Institute for Cybersecurity. *NSL-KDD dataset*. 2009. URL: https://www.unb.ca/cic/datasets/nsl.html (visited on 10/01/2019).

[24] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113. ISSN: 0001-0782.

[25] "Detection Approaches". In: Advances in Information Security. Boston, MA, 2010, pp. 27–53. ISBN: 978-0-387-88771-5. DOI: 10.1007/978-0-387-88771-5_2. URL: https://doi.org/10.1007/978-0-387-88771-5_2.

[26] M. S. Dildar et al. "Effective way to defend the hypervisor attacks in cloud computing". In: *2017 2nd International Conference on Anti-Cyber Crimes (ICACC)*. Mar. 2017, pp. 154–159. DOI: 10.1109/Anti-Cybercrime.2017.7905282.

[27] Wade Edwards et al. *CCNP Complete Study Guide: Exams 642-801, 642-811, 642-821, 642-831*. en. John Wiley & Sons, Feb. 2006.

[28] R. Fagin, R. Kumar, and D. Sivakumar. "Comparing Top k Lists". In: *SIAM Journal on Discrete Mathematics* 17.1 (Jan. 1, 2003), pp. 134–160.

[29] Romain Fontugne et al. "MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking". In: *ACM CoNEXT '10*. Philadelphia, PA, Dec. 2010.

[30] R. F. Fouladi, O. Ermiş, and E. Anarim. "Anomaly-Based DDoS Attack Detection by Using Sparse Coding and Frequency Domain". In: *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. Sept. 2019, pp. 1–6. DOI: 10.1109/PIMRC.2019.8904393.

[31] Sebastián García et al. "An Empirical Comparison of Botnet Detection Methods". In: *Computers & Security* 45 (Sept. 2014), pp. 100–123. DOI: 10.1016/j.cose.2014.05.011.

[32] B. Golden. *Amazon Web Services For Dummies*. –For dummies. Wiley, 2013. ISBN: 9781118651988.

[33] *Google Cloud*. https://cloud.google.com/. [Online; accessed Jan-20-2018].

[34] Shashank Gupta and B. B. Gupta. "Cross-Site scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art". In: *International Journal of System Assurance Engineering and Management* (Sept. 14, 2015), pp. 1–19. ISSN: 0975-6809, 0976-4348.

[35] Barbara Guttman and Edward A. Roback. *An introduction to computer security: the NIST handbook*. DIANE Publishing, 1995.

[36] H. He et al. "A Novel Multimodal-Sequential Approach Based on Multi-View Features for Network Intrusion Detection". In: *IEEE Access* 7 (2019), pp. 183207–183221. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2959131.

[37] R. van Heerden, L. Leenen, and B. Irwin. "Automated classification of computer network attacks". In: *2013 International Conference on Adaptive Science and Technology (ICAST)*. Pretoria, Nov. 2013, pp. 1–7.

[38] S. Hettich and S. D. Bay. *The UCI KDD Archive*. 1999. URL: http://kdd.ics.uci.edu (visited on 10/01/2019).

[39] Y. Hsu et al. "Toward an Online Network Intrusion Detection System Based on Ensemble Learning". In: *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. July 2019, pp. 174–178. DOI: 10.1109/CLOUD.2019.00037.

[40] *IBM Cloud*. https://www.ibm.com/cloud-computing. [Online; accessed Jan-20-2018].

[41] O. Al-Jarrah and A. Arafat. "Network Intrusion Detection System using attack behavior classification". In: *2014 5th International Conference on Information and Communication Systems (ICICS)*. Apr. 2014, pp. 1–6. DOI: 10.1109/IACS.2014.6841978.

[42] Kristopher Kandall. "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems". B.S. and M. Sc Thesis. Boston, MA, USA: Dept. of Electrical Eng. and Computer Science, MIT, June 1999.

[43] M. Kihl et al. "Traffic analysis and characterization of Internet user behavior". In: *2010 International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. 2010 International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT). Moscow, Oct. 2010, pp. 224–231.

[44] Joseph Migga Kizza. *Guide to computer network security*. Second edition. Computer communications and networks. London ; New York: Springer, 2013.

[45] N. Krishnan and A. Salim. "Machine Learning Based Intrusion Detection for Virtualized Infrastructures". In: *2018 International CET Conference on Control, Communication, and Computing (IC4)*. July 2018, pp. 366–371. DOI: 10.1109/CETIC4.2018.8530912.

[46] *Kyoto2006+ dataset*. URL: http://www.takakura.com/Kyoto_data/ (visited on 10/01/2019).

[47] Hung-Jen Liao et al. "Intrusion detection system: A comprehensive review". In: *Journal of Network and Computer Applications* 36.1 (2013), pp. 16–24. ISSN: 1084-8045.

[48] A. Liu and B. Sun. "An Intrusion Detection System Based on a Quantitative Model of Interaction Mode Between Ports". In: *IEEE Access* 7 (2019), pp. 161725–161740. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2951839.

[49] P. Lotfallahtabrizi and Y. Morgan. "A novel host intrusion detection system using neural network". In: *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. Jan. 2018, pp. 124–130. DOI: 10.1109/CCWC.2018.8301663.

[50] "Machine Learning Techniques for Intrusion Detection System: A Review". In: 72 (Feb. 2015), pp. 422–429. ISSN: 19928645. URL: https://search.ebscohost.com/login.aspx?direct=true&db=aci&AN=101323358&lang=es&site=eds-live.

[51] *Microsoft Azure*. https://azure.microsoft.com. [Online; accessed Jan-20-2018].

[52] S.K. Mitra and S.K. Mitra. *Digital Signal Processing: A Computer-based Approach*. Connect, learn, succeed. McGraw-Hill, 2011. ISBN: 978-0-07-128946-7.

[53] Chirag Modi et al. "A survey of intrusion detection techniques in Cloud". In: *Journal of Network and Computer Applications* 36.1 (2013), pp. 42–57. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2012.05.003. URL: http://www.sciencedirect.com/science/article/pii/S1084804512001178.

[54] N. Moustafa and J. Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)". In: *2015 Military Communications and Information Systems Conference (MilCIS)*. Nov. 2015, pp. 1–6. DOI: 10.1109/MilCIS.2015.7348942.

[55] S. Naseer et al. "Enhanced Network Anomaly Detection Based on Deep Neural Networks". In: *IEEE Access* 6 (2018), pp. 48231–48246. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2863036.

[56] Petteri Nevavuori and Tero Kokkonen. "Requirements for Training and Evaluation Dataset of Network and Host Intrusion Detection System". In: *New Knowledge in Information Systems and Technologies*. Ed. by Álvaro Rocha et al. Cham: Springer International Publishing, 2019, pp. 534–546. ISBN: 978-3-030-16184-2.

[57] J. Nikolai and Yong Wang. "Hypervisor-based cloud intrusion detection system". In: *2014 International Conference on Computing, Networking and Communications (ICNC)*. Feb. 2014, pp. 989–993. DOI: 10.1109/ICCNC.2014.6785472.

[58] Catherine Paquet. *Authorized Self-Study Guide Implementing Cisco IOS Network Security (IINS)*. en. Indianapolis: Cisco Press, Apr. 2009. ISBN: 978-1-58705-883-7.

[59] Alvaro Parres-Peredo, Ivan Piza-Davila, and Francisco Cervantes. "MapReduce Approach to Build Network User Profiles with Top-k Rankings for Network Security". In: *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. Paris, France, Feb. 2018, pp. 1–5. DOI: 10.1109/NTMS.2018.8328702.

[60] Alvaro Parres-Peredo, Ivan Piza-Davila, and Francisco Cervantes. "Towards a User Network Profiling for Internal Security using Top-K Rankings Similarity Measures". In: *40th International Conference on Telecommunications and Signal Processing (TSP 2017)*. Barcelona, Spain, July 2017, pp. 16–19.

[61] Alvaro Parres-Peredo, Ivan Piza-Davila, and Francisco Cervantes. "Unexpected-Behavior Detection Using TopK Rankings for Cybersecurity". In: *Applied Sciences* 9.20 (2019). ISSN: 2076-3417. DOI: 10.3390/app9204381. URL: https://www.mdpi.com/2076-3417/9/20/4381.

[62] S. Perera. *Hadoop MapReduce Cookbook*. Community experience distilled. Packt Publishing, Limited, 2013. ISBN: 9781849517294.

[63] P. H. Pwint and T. Shwe. "Network Traffic Anomaly Detection based on Apache Spark". In: *2019 International Conference on Advanced Information Technologies (ICAIT)*. Nov. 2019, pp. 222–226. DOI: 10.1109/AITC.2019.8920897.

[64] Tao Qin et al. "MUCM: multilevel user cluster mining based on behavior profiles for network monitoring". In: *IEEE Systems Journal* 9.4 (Dec. 2015), pp. 1322–1333. ISSN: 1932-8184.

[65] Markus Ring et al. "Flow-based benchmark data sets for intrusion detection". In: *Proceedings of the 16th European Conference on Cyber Warfare and Security (ECCWS)*. ACPI, 2017, pp. 361–369.

[66] D. D. Roy and D. Shin. "Network Intrusion Detection in Smart Grids for Imbalanced Attack Types Using Machine Learning Models". In: *2019 International Conference on Information and Communication Technology Convergence (ICTC)*. Oct. 2019, pp. 576–581.

[67] M. Salem, S. Taheri, and J. S. Yuan. "Anomaly Generation Using Generative Adversarial Networks in Host-Based Intrusion Detection". In: *2018 9th IEEE Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*. Nov. 2018, pp. 683–687. DOI: 10.1109/UEMCON.2018.8796769.

[68] K A Scarfone and P M Mell. *Guide to Intrusion Detection and Prevention Systems (IDPS)*. en. Tech. rep. NIST SP 800-94 Rev1. Gaithersburg, MD: National Institute of Standards and Technology, 2012. URL: https://csrc.nist.gov/publications/detail/sp/800-94/rev-1/draft (visited on 09/09/2019).

[69] Ali Shiravi et al. "Toward developing a systematic approach to generate benchmark datasets for intrusion detection". In: *Computers & Security* 31.3 (May 2012), pp. 357–374. ISSN: 0167-4048.

[70] Raman Singh, Harish Kumar, and R. K. Singla. "An intrusion detection system using network traffic profiling and online sequential extreme learning machine". In: *Expert Systems with Applications* 42.22 (Dec. 2015), pp. 8609–8624. (Visited on 09/14/2015).

[71] William Stallings. *Cryptography and network security: principles and practice*. Seventh edition. Boston: Pearson, 2014.

[72] William Stallings. *Data and Computer Communications*. 10th. Boston. Ma: Pearson, 2014.

[73] M. Stigler. *Beginning Serverless Computing: Developing with Amazon Web Services, Microsoft Azure, and Google Cloud*. Richmond, VA: Apress, 2017. ISBN: 9781484230848.

[74] P. K. Sujatha et al. "A Behavior Based Approach to Host-Level Intrusion Detection Using Self-Organizing Maps". In: *2008 First International Conference on Emerging Trends in Engineering and Technology*. July 2008, pp. 1267–1271. DOI: 10.1109/ICETET.2008.190.

[75] *The CAIDA Anonymized Internet Traces 2011 Dataset*. URL: https://www.caida.org/data/passive/passive_2011_dataset.xml (visited on 10/01/2019).

[76] Y. Tsuda et al. "A Lightweight Host-Based Intrusion Detection Based on Process Generation Patterns". In: *2018 13th Asia Joint Conference on Information Security (AsiaJCIS)*. Aug. 2018, pp. 102–108. DOI: 10.1109/AsiaJCIS.2018.00025.

[77] S. Vij and A. Jain. "Smartphone nabbing: Analysis of intrusion detection and prevention systems". In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. Mar. 2016, pp. 2209–2214.

[78] Y. Wadia. *AWS Administration - The Definitive Guide: Design, build, and manage your infrastructure on Amazon Web Services, 2nd Edition*. Packt Publishing, 2018. ISBN: 9781788477178.

[79] William Webber, Alistair Moffat, and Justin Zobel. "A similarity measure for indefinite rankings". In: *ACM Trans. Inf. Syst.* 28.4 (Nov. 2010), 20:1–20:38. ISSN: 1046-8188. DOI: 10.1145/1852102.1852106. URL: http://doi.acm.org/10.1145/1852102.1852106 (visited on 10/11/2016).

[80] *What is AWS*. https://aws.amazon.com/what-is-aws. [Online; accessed Jan-20-2018].

[81] J. Xing et al. "AsIDPS: Auto-Scaling Intrusion Detection and Prevention System for Cloud". In: *2018 25th International Conference on Telecommunications (ICT)*. June 2018, pp. 207–212. DOI: 10.1109/ICT.2018.8464855.

[82] Kuai Xu, Feng Wang, and Lin Gu. “Behavior analysis of internet traffic via bipartite graphs and one-mode projections”. In: *IEEE ACM Transactions on Networking* 22.3 (June 2014), pp. 931–942.

[83] S. Yang. “Research on Network Behavior Anomaly Analysis Based on Bidirectional LSTM”. In: *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. Mar. 2019, pp. 798–802. DOI: 10.1109/ITNEC.2019.8729475.

Index of Authors

Subject Index