# Security in the Cloud : an anomaly-based detection framework for the insider threats

Pamela Carvallo

université PARIS-SACLAY

TELECOM SudParis

**NNT : 2018SACLL008**

# Thèse de doctorat de L'Université Paris-Saclay preparée à Télécom SudParis

Ecole Doctorale N° 580
Sciences et Technologies de l'Information et de la Communication

Spécialité de doctorat Informatique

Par

**Mme. Pamela Carvallo**

## Sécurité dans le cloud : Framework de détection de menaces internes basé sur l'analyse d'anomalies

**Thèse présentée et soutenue à Évry, le 17/12/2018 :**

**Composition du Jury :**

| | | |
|---|---|---|
| M. MAAG, Stéphane | Professeur, Télécom SudParis | Directeur de thèse |
| Mme. CUPPENS, Nora | Directrice de recherche, IMT Atlantique | Rapportrice |
| Mme. WAESELYNCK, Hélène | Directrice de recherche, LAAS-CNRS | Rapportrice |
| Mme. KUSHIK, Natalia | Maître de conférences, Télécom SudParis | Co-encadrante |
| Mme. CAVALLI, Ana Rosa | Professeur Emérite, Télécom SudParis | Examinatrice |
| M. MELLOUK, Abdelhamid | Professeur, Université de Créteil | Examinateur |
| M. POIZAT, Pascal | Professeur, Université Paris Nanterre | Examinateur |
| M. MALLOULI, Wissam | Ingénieur de recherche, Montimage | Examinateur |

*"There is a theory which states that if ever anyone discovers exactly what the Universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable. There is another theory which states that this has already happened." – Douglas Adams.*

# Acknowledgments

First of all, I would like to express my sincere gratitude to my thesis director Stéphane Maag and Ana Cavalli for giving me the opportunity to do research and materialize the work presented in this document.

Also, the confidence and opportunity given by Edgardo Montes de Oca, whom also put his effort and confidence by allowing me to work in the European Project MUSA by which my thesis was founded. I am very grateful for the opportunity and the help he and all my colleagues in Montimage gave me to collaborate with partners across Europe while, at the same time, develop the work for the thesis.

Aside from the support of my supervisors, I would also like to convey my sincere gratefulness to Natalia Kushik, co-director of this thesis, for her continuous support and interest in the work I developed. She always supported, evaluated and expanded my work and research questions with fulfilling discussions. This work has been possible by the enrichment of our collaboration.

In addition, my colleagues at the research group in Télécom SudParis, whom I shared and discussed several ideas and personal guidance. I thank Jorge López for his selfless support in this personal project. Diego Rivera, whom I met as a PhD student and now works aside from me in Montimage, for his sharp view and support (along with help in French writing — you grammar freak).

In a personal level, I would like to thank with all my heart to all the people that stood and shared this journey with me. My devoted mom and brother, who have believed in me and unconditionally encouraged me in all my life initiatives. Also, my father, whom I presume would have been very proud to see this achievement. My friend Silvana who gave me perspective in both rough and passionate moments throughout my work. Likewise, my friends at my new home in Paris, Mario, and Andrea, for their philosophical discussions (they are both PhDs, if you know what I mean) and Anaïs for her sincere friendship.

Last but not least, to all the intellectual scientific mentors, writers and artists that have and continue inspiring me to do research, to aim at understanding and looking at life with hungry eyes.

# Contents

# List of Figures

# List of Tables

Chapter 1

# Introduction

**Contents**



## 1.1 Motivation

Cloud computing opens new possibilities for more flexible and efficient services. However, one of the issues of migrating to the cloud is that it involves a third-party implementation and enforcement of security policies [110]. In such environments, many security aspects must be faced, including risk management, data privacy and isolation, security-by-design applications, vulnerability scans, among others. Besides preventive solutions (e.g., encryption, firewalls), it also becomes necessary to have a system that interrelates all monitored security mechanisms from different points of observation. On the other hand, new attacks emerge every day, and thus, threat detection systems start playing a key role in security schemes, identifying possible attacks.

According to recent reports [31], 90% percent of organizations feel vulnerable to insider attacks. The main enabling risk factors include too many users with excessive access privileges (37%), an increasing number of devices with access to sensitive data (36%), and the increasing complexity of information technology (35%). Also, a majority of 53% confirmed insider attacks against their organization in the previous 12 months (typically less than five attacks). Twenty-seven percent of organizations say insider attacks have become more frequent. Additionally, organizations are shifting their focus on detection of insider threats (64%), followed by deterrence methods (58%) and analysis and post-breach forensics (49%). The use of user behavior monitoring is accelerating; 94% of organizations deploy some method of monitoring users and 93% monitor access to sensitive data. Also, technical reports from the industry [25], [38] state that threat researchers also examined data ex-filtration trends, using a machine-learning algorithm to profile 150,000 users in 34 countries, all using cloud service providers, from January to

June 2017. After profiling users for six months, researchers spent 1.5 months studying abnormalities, flagging 0.5% of users for suspicious downloads. These users downloaded, in total, more than 3.9 million documents from corporate cloud systems, or an average of 5200 documents per user during the 1.5-month period. Of the suspicious downloads, 62% occurred outside of normal work hours; 40% took place on weekends. By benefiting from machine learning techniques, algorithms can learn and improve themselves by studying high volumes of available data, but more efficient learning procedures are needed, in order to increase these detection rates.

More specifically, cloud-based threat detection techniques are commonly divided into three groups, namely pattern-based, behavior-based and hybrid (first two combined). In the first case, attacks are described as rules or expressions in related grammar (signatures), and the newly collected data of the system under test are verified with respect to the set of such signatures. For behavior-based techniques, "normal" system actions are somehow defined, and the monitoring system can later serve for concluding if the monitored environment is differing from the defined behavior. In this case, different statistical models, as well as self-learning techniques, can be effectively utilized. Nevertheless, some proposed algorithms commonly suffer from high false-positive detection rates, encouraging the use of hybrid approaches by companies and academic institutions.

Unsupervised learning is arguably more typical of human and animal learning. It is also more widely applicable than supervised learning since it does not require a human expert to label the data manually. Labeled data are not only expensive to acquire, but it also could dismiss the information [92].

The advantage of such neurally-inspired clustering approaches lies in their ability to learn the representation of a feature space without supervision. A further interesting property is the fact that they typically perform dimensionality reduction. GNG represents an extension of the Neural Gas (NG) algorithm in which the number of neurons is not fixed *a-priori* as in NG, but grows over time. This feature is especially interesting in such clustering tasks where the number of clusters is previously unknown.

This research describes the difficulty in modeling and detecting insider threat considering three factors. First, it is a *low-rate problem* because detected incidents are relatively rare events. Second, it is a *misperceived problem* because security audits are in place focusing mostly on external attacks. Third, it is a *high-impact problem* because unlike an external threat, insider threat tends to go undetected and can involve long-term malicious activities. An example of this is that most of the insider attacks were only discovered through manual (non-automated) detection of an irregularity or failure of an information system [74].

Moreover, this threat is more dangerous in a cloud environment than in a traditional Information technology (IT) infrastructure because the insider may gain access to data from other Cloud Internet Service Client (CSC) hosted by the Cloud Internet Service Provider (CSP). Studies conducted by Verizon [9], [136] indicate that internal breaches continue to be much more damaging than other sources of attack. Besides, they indicate that there are more insider breaches this year than ever before. However, companies focus mostly on preventing external attacks.

Although several research works have addressed relevant indicators when trying to detect malicious insider threats [26], [29], [33], [65], [66], [95]. However, to the best of our knowledge, very few publications [26], [33], [66] discuss their implications in a cloud environment.

Furthermore, when aiming to detect this malicious activity, the confidentiality and privacy of CSPs and CSCs concerning their internal organization and policies, create barriers for the collection and utilization of data for research purposes. Moreover, despite the predictions and

possible creative attacks presented by researchers, there is very little evidence of actual events involving the type of insider described in the Cloud Security Alliance (CSA)'s [27] document. Additionally, addressing malicious activities also presents challenges since they vary according to the cloud service model, CSC characteristics such as services used, job types and organizational hierarchy.

When tackling this threat, existing solutions aim at performing a detection approach considering hard-task implementations or information assumptions. Such is the case of detection schema dealing with broad access to data sources (e.g., logs) and even, labeled data. Also, the studied methods are typically implemented in batch mode, and thus they cannot be easily extended to anomaly detection problems with streaming data or online settings.

Lastly, they treat the employee's behavior as a one-class problem, i.e., new examples are classified as either belonging to the target class or as an anomaly or novelty. This may differ from reality, where user's behavior may vary in time and is dependent on daily changing activities, varying policies and roles.

## 1.2 Problem statement

The research goal of the present work is to determine whether user behaviors, network traffic and cloud-based attributes can be used as an indicator to identify insider threats. If there is a relationship between malicious users and their behaviors that are different from normal users, it is possible to identify insider threats using these predictive indicators.

Therefore, the goal of this work is then formulated as follows:

> ***Given a monitoring engine that outputs unlabeled employees' activity traces, and a collection of their past online activity, we want to detect employees behaving abnormally concerning their past actions, with high accuracy and a low false positive rate, at a current time step and in an unsupervised approach. This goal is pursued by the proposal and deployment of an anomaly detection system based on the extension of a set of existing techniques.***

Accordingly, the following research questions arise:

- Data treatment: Which data should be considered to analyze the user's anomaly?
- Which algorithms are best suited for detecting the insider threat?
- How these existing algorithms should be adapted, if necessary?

We note that this thesis does not aim to find one universal best technique. It is not plausible because the experiments will be conducted with a specific type of data, namely, network traces from the OpenStack cloud implementation. This thesis will explore only a few ways to measure the detection performance between different techniques. Nevertheless, the proposed solution is conceived by means of an extensive modular approach, therefore tackling the future reduction augmentation of more monitoring attributes.

## 1.3 Contributions

The main contribution of this work is the analysis, modeling and validation for the insider threat detection. Moreover, this work presents a fundamental methodology for assessing a detection from a behavioral data-driven point of view, exposing data-related detection challenges, implementations trade-offs with respect to detection performance, finally giving a broader view of considering factors at the moment of dealing with such a complex threat scenario.

The first contribution consists of a literature review and a study providing a closer relationship between existing techniques and tools for effective threat detection in cloud environments. Also, an evaluation is performed of a group of existing detection techniques concerning cloud computing principles and security challenges nowadays. Finally, an experimental study is presented of some commonly utilized datasets and their association with threats in the last five years.

We assessed the following methodology: we covered the topics of detection systems and detection techniques in the cloud. The selection of works derived in both a systematic review of the detection architectures and the second in a detailed examination of the detection techniques. As existing detection techniques tend to target specific threats (or their groups), an experimental evaluation of the applicability was also presented, for known detection approaches against non-targeted threat groups.

The second contribution is focused on the derivation and validation of the dataset for cloud-based malicious insider threat. Consequently, it addresses a dataset generation methodology that takes into account various issues, including statistical analysis as well as the creation of cloud-related user scenarios. The contribution is motivated by the complexity of the problem itself as well as by the absence of open, realistic cloud-based datasets. Also, it presents a dataset validation criteria based on a set of predefined rules that include statistical evaluation. Finally, there is a design and presentation of a cloud-based proof-of-concept with malicious insider attacks.

The third and main contribution of this work tackles the detection of the threat mentioned above through an anomaly detection framework. The latter addresses the insider threat employing machine learning techniques, due to their ability to look into data and look for patterns which can be learned and improved adaptively in time. In detail, motivated by the above-presented insider threat detection characteristics, the particular technique in use considers an unsupervised and online approach, allowing the solution to identify different user behaviors for present and new data, in a stream-fashion.

Moreover, this contribution is part of a modular design, capable of dealing with several data input sources, from different nature, namely user actions using text representation, network traffic and cloud-related attributes. For the first set of features, and given the fact that generally, machine learning algorithms rely on numeric representations, further analysis of the different ways of text representation is also examined.

Furthermore, the novel framework for the insider threat makes efficient use of domain knowledge given by known enterprise policies and security experts (e.g., cloud administrator) utilizing them as additional contextual information. This idea is explored both as attributes used by the detection algorithm automatically, and also in virtue of labeling/identifying anomalies that could be treacherous.

Finally, to illustrate the generality of the proposed framework, experimental evaluation is applied to different insider threat case scenarios. This benchmark work evaluates the relevance of the different data sources, along with the context-based anomaly detection approach.

## 1.4 Publications

The aforementioned contributions have been published in different conference proceedings and are part of several citations by academic peers. A complete list of the publications related with the present work is the following:

1. [21]: Carvallo P., Cavalli A.R., Kushik N. Automatic derivation and validation of a cloud dataset for insider threat detection. ICSOFT 2017 : 12th International Conference on Software Technologies, Jul 2017, Madrid, Spain. Scitepress, Proceedings ICSOFT 2017 : 12th International Conference on Software Technologies, pp.480 - 487, 2017.

2. [18]: Carvallo P., Cavalli A.R., Kushik N. (2018) A Study of Threat Detection Systems and Techniques in the Cloud. In: Cuppens N., Cuppens F., Lanet JL., Legay A., Garcia-Alfaro J. (eds) Risks and Security of Internet and Systems. CRiSIS 2017. Lecture Notes in Computer Science, vol 10694. Springer, Cham.

In addition, there have been contributions to the European MUSA project (Multi-Cloud Secure Applications) [93], within the framework of the H2020 program, the main objective of which is to support the security lifecycle management of distributed applications, heterogeneous cloud resources.

3. [19]: Carvallo P., Cavalli A.R., Mallouli W. (2018) A Platform for Security Monitoring of Multi-cloud Applications. In: Petrenko A., Voronkov A. (eds) Perspectives of System Informatics. PSI 2017. Lecture Notes in Computer Science, vol 10742. Springer, Cham.

4. [20]: Carvallo P., Cavalli A.R., Mallouli W., Rios E. (2017) Multi-cloud Applications Security Monitoring. In: Au M., Castiglione A., Choo KK., Palmieri F., Li KC. (eds) Green, Pervasive, and Cloud Computing. GPC 2017. Lecture Notes in Computer Science, vol 10232. Springer, Cham.

Lastly, we are currently preparing a paper to the "Computers & Security" Journal, where the insider threat detection framework and related experimental findings form the main contributions.

## 1.5 Organization of the thesis

The thesis is organized in seven chapters. A brief description of the chapter contents are below given.

Chapter 1: **Introduction**. It provides a general background, context and motivations. The main objectives and research questions are stated, as well as the thesis structure, main contributions and publications.

Chapter 2: **State of the Art**. It discusses the related concepts of the main security challenges in a cloud scenario, along with the most relevant monitoring mechanisms to protect its assets from threats. We also provide a study of the most important threats followed by a deep analysis and interrelation of the specific techniques to detect each one of them. Additionally, this chapter presents the most utilized techniques to tackle the insider threat.

Chapter 3: **Insider Threat Model**. It presents the simulation engine that characterizes the insider threat intended to detect. It provides different configuration possibilities, showing the adeptness of its usage.

Chapter 4: **Anomaly-based Detection Framework**. It describes the proposed detection model to assess the different insider threat scenarios from the previous Chapter 3.

Chapter 5: **Implementation**. It presents the implementation details for the monitoring and detection technique in a cloud-based environment.

Chapter 6: **Experimental evaluation**. It demonstrates the effectiveness of the proposed detection approach through extensive experimental evaluation.

Chapter 7: **Conclusion**. It provides some answers to the research questions described in Chapter 1, which are based on the results obtained from this research. It also points towards future research directions.

Chapter 2

# State of the Art

**Contents**



## 2.1 Introduction

This chapter presents a study of existing threat detection techniques in cloud computing, together with an experimental evaluation of a subset of them. We consider the threats defined in the Cloud Security Alliance (CSA) report as well as the techniques for their detection, starting from classical signature-based approaches and finishing with recent machine learning based techniques.

As the topic of providing security in the cloud remains essential, it is worth mentioning that the state of the art presented in this work is not the first covering this subject. However, existing works (e.g., [63], [69], [71], [88], [100], [104]) mostly focus on either analyzing system requirements and cloud security gaps, or describing detection techniques along with some attacks.

It is required then to combine both approaches to provide a broad view of the state of the art of the problem. Below, we briefly sketch some existing works summarizing security issues in the cloud and discuss the motivation for expanding the research on this field.

## 2.2 Preliminaries

### 2.2.1 Security issues in the cloud

According to the National Institute of Standards and Technology [86], cloud computing is a model for enabling ubiquitous, elastic, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services). Its service models, known as Software as a Service (SaaS), Platform as a Service (PaaS) and IaaS have specific and shared security challenges. The first provides a CSC the capability to use applications running on a cloud infrastructure by a CSP. PaaS provides the CSC with tools to deploy their applications on top of the cloud infrastructure. The latter gives provision to the CSC in processing, storage, networks, and other fundamental computing resources where the consumer can deploy and run arbitrary software, which can include operating systems and applications. Security challenges reside in the coexistence since PaaS, as well as SaaS services, are hosted on top of IaaS. Therefore, any threat in IaaS can impact the security of both, PaaS and SaaS, and vice versa [57].

We further enumerate critical cloud aspects trying to provide the explanations how these elements can influence the design of a cloud-based threat detection system. In particular, we consider:

- **Virtualized environment** Brings elasticity by allowing multiple Virtual Machine (VM) management and pooling in the same physical resources.
- **Multi-tenancy** Enables the use of a single resource by multiple customers that may or may not belong to the same organization.
- **Data life cycle** Defines no fixed infrastructure and security boundaries on applications and data on the cloud.
- **Network dynamics** Concerns non-linear, non-stationary and complex dynamical characteristics of the network flows.
- **Access** Takes into account the fact that data are transmitted using the Internet and may require credentials, authentication, identity management and anonymization.

### 2.2.2 Overview of cloud-related threats

Following the European Network and Information Security Agency [36] we consider a *threat* as an event that can exploit a vulnerability, intentionally or accidentally, and obtain, damage, or destroy an asset. An *attack* is a sequence of components and interfaces that a threat actor or a condition can use to achieve a threat against an asset. The threat actor or actors gain access to the assets via attack vectors and vulnerabilities present in the technology components that host or provide direct access to the targeted assets. *Threat detection systems* are deployed in cloud environments with the intent to prevent, address and mitigate the attacks pursued by the threat actors, thereby protecting the assets.

Common threat guidelines have been proposed reflecting the current concerns among experts [32], [36], [41], [57], resulting in data threats such as breaches or losses, account hijacking, insecure application programming interfaces, DoS, malicious insiders, abuse of cloud services and shared technology. These have been previously reviewed in [129], together with the relevant vulnerabilities and countermeasures analysis [57]. From the preceding sources, we list below the considered group of threats in this work, along with two threat groups gathered from the aforementioned studies: Other attacks (corresponding to known attack patterns from network datasets, such as port scan) and Malware (e.g., Kelihos and Zeus), illustrated in Figure 2.2.

- **Data-related threats** Treated as the top threats among industry experts [32]. A *data breach* is an incident in which protected or confidential information is released, viewed, stolen or processed by an entity not authorized to do so. It concerns IaaS, PaaS and SaaS as they all keep sensitive data.

- **Account Hijacking** Specified as a process in which an individual or organization's cloud account is stolen or hijacked by an attacker. This threat is relevant to cloud architectures since attackers can often access critical areas of deployed cloud computing services, allowing them to compromise the confidentiality, integrity and availability of IaaS, PaaS and SaaS services.

- **Malicious Insider** Defined as a threat to an organization occasioned by a current or former employee, contractor, or another business partner who has or had authorized access to an organization network, system, or data. This action intentionally exceeded or misused the access in a manner that negatively affected the confidentiality, integrity, or availability of the organization information or information systems.

- **Denial of Service** Meant to prevent components from being available in a cloud environment; that concerns, for example addressing to APIs for SaaS outage or specific Distributed Denial of Service (DDoS) at the infrastructural layer [4].

- **Shared Technology threats** Existent in all delivery models, including multi-tenant architectures (IaaS), re-deployable platforms (PaaS), or multi-customer applications (SaaS) [32].

## 2.3 Cloud threat detection systems

Threat detection systems usually correspond to a hardware device or software application that monitors an activity (e.g., from network, VM host, user) for malicious policy violations. Previous works (e.g., [104], [131]) have stated several features of detection systems; among those, fault-tolerance, real-time execution, self-monitoring, minimum operational, interoperability, self-adaptiveness, scalability. A multi-criteria analysis of Intrusion Detection System (IDS) was presented in [146], following these and other cloud computing requirements such as performance and availability along with CSA-inspired criteria, such as service level expectations, secured and encrypted communication channels, detection methods used and their accuracy, among others.

System architectures may vary if they are distributed, centralized, agent-based [60] or collaborative; the positioning of various observation points also defines different types of architectures. The monitoring layers can be classified as follows:

1. Network-based monitor activity of network traffic —mostly Internet Protocol (IP) and transport layer;

2. Host-based monitor application or service activities operating on top of VM's operative systems;

3. Hypervisor-based monitor virtual machine introspection to gather system-specific features (e.g., process list, threats count, number of open ports);

4. Cross layer-based monitor in the form of any combination of the previously mentioned.

In general, *data collection and preparation* are performed through a sensor or existing dataset. This information works as an input for the *data analysis and detection*, which corresponds to the module of the algorithms implemented to detect suspicious activities, detailed in the following sections.

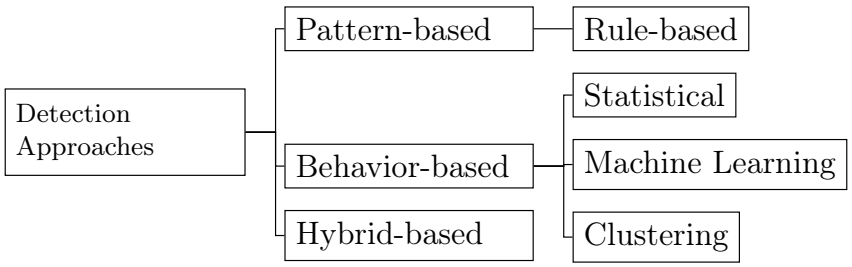



Figure 2.1 – Classification of threat detection techniques

### 2.3.1 Pattern-based approach and related techniques

Also known as "signature-based", "knowledge-based" or "misuse-based", this approach operates over a set of rules that define a threat pattern or a known authorized pattern. They are known to have a high level of accuracy [121], but are limited to only known rules and attacks. Therefore, pattern-based techniques cannot detect variants of known or unknown attacks. Moreover, keeping signature or knowledge databases updated may be a hard task.

Latest research focuses on facilitating to cloud administrators the determination of new attack patterns by updating signature databases more efficiently. To assess this automatic and offline analysis, Hamdi et al. [55] proposed Inductive Logic Programming, while Huang et al. [58] used Growing Hierarchical Self Organizing Maps (GHSOM) for the characterization of attack signatures. Other techniques that we further discuss are grouped as so-called *rule-based*.

#### 2.3.1.1 Rule-based

For known or variants of known attacks, rule-based context methods have been considered in a number of works.

**Watermarking** was studied for *data breaches* detection by Garkoti et al. [48]. Threats may occur in any stage of the data cycle (Section 2.2.1) and digital watermarking is a reviewed technique for detecting data tampering. Specifically, the authors introduced spatial domain watermarking, encryption and logging modules for clinical data. Concerning insider threats and

further potential data-related threats, Kumar et al. [75] considered a method related to the well-known *Bell-LaPadula* model, which aims to determine the organization employee who leaked the data. This model is built on the concept of subjects and objects (i.e., a file). They define levels where subjects have access to objects following security policies. Various cryptographic and watermarking techniques are later applied to identify the internal user involved in the leakage.

**Fingerprinting** was considered for *malicious insider* threat detection by Gupta et al. [54] through the analysis of commonly used programs by a VM. They assumed that the signature of frequent executions remains reasonably constant and detects malicious modifications of the system call sequences executed from the VM to the hypervisor.

**Provable Data Possession (PDP)** formalized in [7], is related to *data losses* preserving the data integrity. Basically, a CSC uploads data for storage and keeps meta-data for later verification. The classical idea behind this technique can only be applied to static (or append-only) files. Hence, Erway et al. [39] presented a framework based on Dynamic Provable Data Possession, which extends the PDP traditional approach. It supports provable updates to the stored data, using a new version of authenticated dictionaries based on rank information.

**Sequence alignment** commonly used in bioinformatics, was proposed by Kholidy et al. [70] to detect *account or service hijacking* threats, specifically for masquerade attacks. They introduced Heuristic Semi-Global Alignment algorithm, which tests matching patterns of user's session sequences (e.g., mouse movements, system calls, opened windows titles, written commands, opened file names) with the previously stored arrays.

**Dependency Graphs** were proposed by Yaseen et al. [140]. Based on applying knowledge and dependency graphs one can detect and predict *malicious insiders* in relational databases. The authors considered the network overhead and system performance for variables, including the number of queries per insider, the number of insiders and percentage of accessibility for data items in relational databases.

### 2.3.2 Behavior-based approach and related techniques

Also known as anomaly-based detection, this approach involves the collection of data in order to construct a model of *normal* behavior and then to test newly observed behaviors against potential anomalies. As this is a sophisticated task, some works have proposed a mixed approach (e.g., [68], [101], [139]) where the following statistical and machine learning methods are combined. We have differentiated existing techniques in statistical, machine learning-based and clustering techniques. We hereafter assume that statistical methods mostly use specific formulas or functions to compute the corresponding characteristics of the data attributes; machine learning, on the other hand, "works" when such functions cannot be derived, and thus, it utilizes more complex relationships between the data for further threat prediction.

#### 2.3.2.1 Statistical

These approaches are in general predefined by a threshold in order to identify anomalies. As an example one can consider a type of Denial of Service (DoS) — Economic Denial of Sustainability (EDoS) — issued by [8], where the authors compared user demands against thresholds of

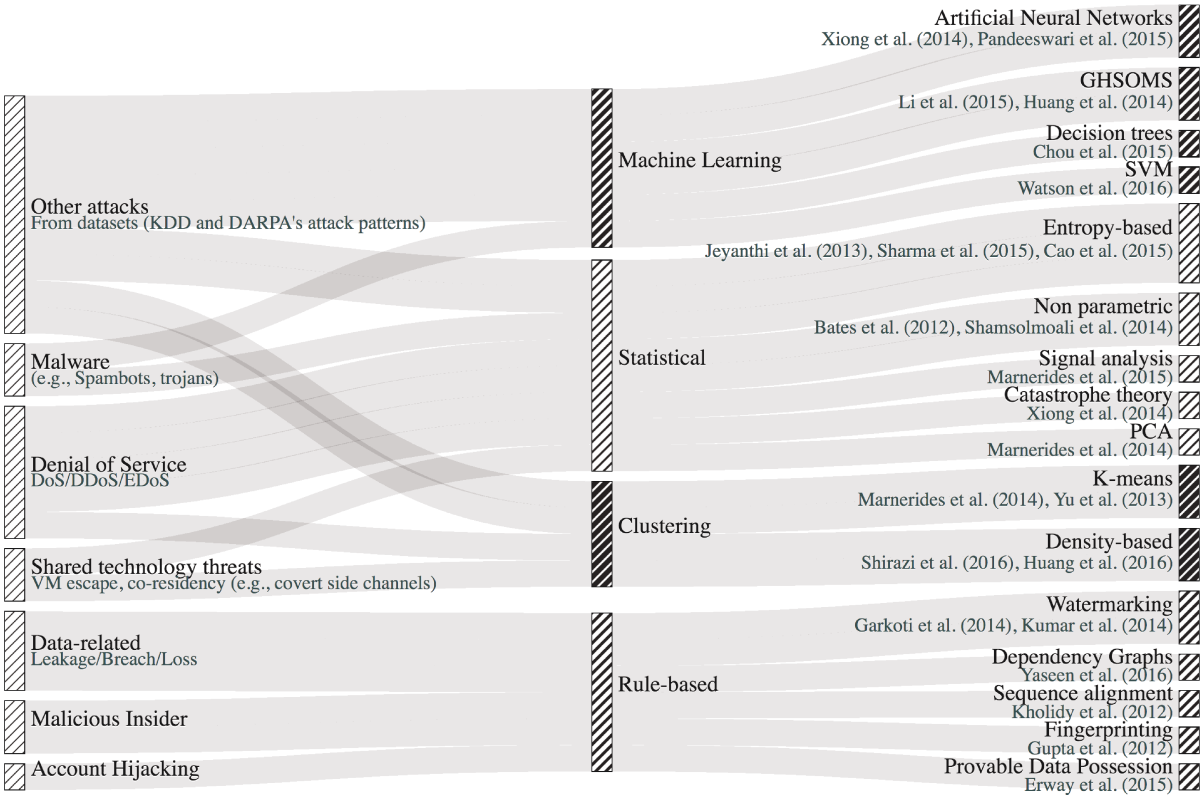



Figure 2.2 – Relationship between the threats and reviewed detection techniques

duration parameters as the maximum number of requests beyond when the auto-scaling feature is activated.

**Non-parametric** techniques take place when the system observes the activity of subjects in terms of statistical distribution and creates profiles which represent their behaviors for later similarity comparison. Bates et al. [10] addressed covert-side channel threats by detecting network flow's watermarking. In particular, they modeled packet arrivals by a Poisson distribution and applied the non-parametric Kolmogorov-Smirnov test. DoS was studied by Shamsolmoali et al. [117] in a two-stage detection: extracting the *Time-to-Live* values from incoming IP packets, computing the number of hops the packet had traveled and using Jense-Shannon divergence searching for anomaly in the *normal* trained database.

**Entropy-based** techniques focus on measuring the uncertainty or randomness associated with a variable. For network flows, comparing the rate of entropy of some packet header fields with other samples of the same nature provides a mechanism for detecting changes in the randomness. For DDoS attack detection, Jeyanthi et al. [61] proposed a cross-layer implementation where the first component analyzes incoming traffic rate and is handed to a Hellinger Distance-based entropy profiler in case it exceeded a threshold. Sharma et al. [118] utilized incoming network packets and studied the entropy of source MAC address also with a threshold-based analysis. Same threat concern and approach were followed in [16] where the authors investigated the VM CPU usage and network interfaces, arguing on the fact that malicious VMs share similar attack

patterns.

**Principal Component Analysis (PCA)** was used by Marnerides et al. [84] for DoS and netscan detection, not only for reducing datasets dimensionality but also to separate the normal data from anomalous.

**Signal analysis** such as Ensemble Empirical Mode Decomposition (E-EMD) was presented in [83]. The authors proposed a data-driven method for malware, motivated by the fact that the algorithm can sufficiently decompose data as signals and describe clouds' non-linear and non-stationary network traffic and hypervisor information.

**Catastrophe theory** studies the way systems respond to the continuous modifications from the variables that control them, producing sudden changes from one system state to another (e.g., from *normal* state to *anomalous*). Xiong et al. [139] introduced a catastrophe function to describe network traffic anomalies in cloud communications.

#### 2.3.2.2 Machine learning-based

These methods allow improving the performance of their objective by learning from previous results. The results summarized in this subsection are illustrated in Figure 2.2, where we group these techniques with their underlying models for detecting security threats in the cloud.

**Decision trees** are used in [24], where they preprocessed unlabeled data with an unsupervised clustering algorithm. After labeling, a model based in incremental tree inducer is trained, therefore updating itself.

**SVM** technique for cloud threat detection was proposed by Watson et al. [134]. The authors studied an online novelty implementation of a supervised one-class SVM algorithm, an extension of traditional two-class SVM which outputs either a known class (VM *normal* behavior) or unknown classes to the classifier, for each particular input vector.

**Artificial Neural Network (ANN)** expose their accuracy based on the configuration of their hidden layers and training phase. Pandeeswari et al. [101] preprocessed hypervisor attributes with Fuzzy C-Means clustering and utilized feed-forward neural networks with back-propagation algorithm for each of them. They later combined the results of the ANNs with a fuzzy aggregation module. A Synergetic Neural Network (SNN) was addressed by Xiong et al. [139], given the dynamics of the network's traffic. Their argument relied on the fact that under some situations, the changing trend of the cloud-based network traffic is only determined by a few primary factors and less contribution of others.

**Self Organizing Map (SOM)** techniques were also addressed by Li et al. [80], by proposing a cluster system that identified Nmap malicious behaviors in VMs through system call distributions in order to derive rules for SVM detection.

#### 2.3.2.3 Clustering

These techniques are utilized under the assumption that *normal* data instances lie distance-wise closer to a given centroid of a cluster, whereas *anomalous* data points are recognized due to their much longer distance (e.g., K-means). Density-based approaches rely on the fact that *normal* data instances belong to large and dense clusters, while *anomalies* either belong to small or sparse clusters.

The idea is that data points in the same cluster are as similar as possible, and consequently, data points in different clusters differ to a high degree. One of the major challenges regarding clustering is to come up with a function that describes similarity between data points, known as similarity function. Algorithms for clustering can be divided into two main categories, namely, partitional and hierarchical algorithms.

**K-means** technique was followed by Marnerides et al. [84], while showing the clustering method is directly affected by live-migrations. In this testbed, they detected DoS and netscan threats successfully when arose, but also achieved high scores when only migration and normal traffic occurred. Additionally, it was utilized for detecting *shared technologies threat*, as seen in Figure 2.2. For example, in [144], the authors combined a two-stage detection mode based on statistical similarity tests from the *cache miss times* from hosts, Central Processing Unit (CPU) and memory utilization collected from VMs, for later clustering.

**Density-based** technique was proposed by Shirazi et al. [121] where they divided all measured variables into clusters and evaluated mean and standard deviation, based on the Euclidean distance threshold. The same clustering idea was used with the dimension reasoning technique (based on Local Outlier Factor) for memory leakage and malicious port scan, by Huang et al. [59].

### 2.3.3 Hybrid-based approach and related techniques

Depending on the architecture and a set of threats to be detected, the use of techniques in cloud architecture can require a hybrid approach.

While signature-based approach is more rigorous in its detection, behavior-based methodology is able to "learn" new threats. Therefore, the combination of previously mentioned approaches in Sections 2.3.1 and 2.3.2 may reach a more extensive and accurate detection. As an example, Modi et Patel. [87], used SNORT [22] for signature-based detection, whereas for anomaly-based detection they focused on Bayesian, associative and decision tree classifiers. Some of the studies addressing both approaches can be found in [70], [87], [117], [145].

## 2.4 Discussion

The classification described in the previous sections shows that signature-based methods commonly relate to content-based detection techniques since they test known patterns or accepted actions. Data-related, malicious insider and account hijacking threats (e.g., confidential documents leakage, allowed user behaviors) are mostly studied in this category. For a visual representation of these dependencies, links are depicted in Figure 2.2, where rule-based groups only share relations with the previously mentioned threats and are not associated with DoS, shared technology threats or malware.

As usual, detection techniques discussed above have their unique strengths and weaknesses. From the results of this study (Figure 2.2 and Section 2.3.2), one can see that the most reviewed group of techniques are the statistical-based and machine learning, often utilized for network traffic and DoS detection. The first relies on the assumption that *normal* data instances fit a statistical model and *anomalies* are compared to this model through inference tests, which may be unhandy for diverse data. Entropy-based techniques offer a deeper examination as they consider the irregularities in the information content of the data being collected. Machine learning algorithms are also efficient due to their self-learning capability. Other approaches such as clustering, add an interesting enhancement since they automatically create and label clusters for future classification.

Also, evaluating the effectiveness of a given detection technique against a particular threat (or a group of them) is mainly performed through corresponding experimentation. For that reason, it is highly relevant building proper datasets that contain heterogeneous *normal* and *abnormal* realistic behaviors with a broad spectrum of threat patterns. Consequently, it may be intuitive to handle combined datasets, as mentioned in the previously cross layer-based system. This implies selecting relevant features, focusing on minimizing used bandwidth during monitoring, improving detection performance and removing redundant data, while keeping lower computational complexity (e.g., machine learning techniques, where the time taken to train the classifier is dataset size dependent).

Literature regarding this matter has used self-generated test-beds [59], [61], [84], [121], [134] while others relied on the well-known datasets: KDD [24], [101], [117] and DARPA [24], [139]. These last two correspond to the group of threats with more references in Figure 2.2. However, they suffer from several deficiencies for testing in cloud environments as they do not include behaviors such as stated in Section 2.2.1. Accordingly, a dataset containing new malware patterns was used and is presented in the next section.

## 2.5 Experimental evaluation of studied techniques and known threats

The aim of the experimental evaluation was to identify the concrete challenges of the thesis and to study the missing relation between some threats and a technique of each group. These connections were formerly determined by the reviewed publications, where Figure 2.2 graphically illustrates which set of techniques has been utilized for detecting different threat categories (from Section 2.2.2). We conducted experiments to estimate the effectiveness of these techniques against other threat types, therefore contributing by adding new links to our study.

To the best of our knowledge, such experiments were not performed before in the time of study, i.e., 2016, for the following detection algorithms against the utilized dataset: SVM, MLP feed-forward Neural Network, and LSTM Recurrent Neural Network, K-means and entropy-based. We have selected one technique of each group to perform a more exhaustive analysis. The first is commonly used as benchmark experiments outperforming in most cases ([71], [104]); hence, it was of our interest to see how it performs for the chosen dataset's attacks. The second and third techniques enhance the dynamic classification requirement, presented in Section 2.3.

Moreover, the study targeted the usage of techniques with self-learning capabilities (i.e., that handle new data after the training phase). Following this idea, MLP and LSTM present relevant characteristics. The last two techniques belong to the clustering and statistical categories, respectively. K-means is a learning algorithm that groups attribute vectors in clusters, based on

Table 2.1 – Average detection performance for SVM, MLP, LSTM, K-means and entropy-based techniques.

| Metric (%) | SVM | MLP | LSTM | K-means | Entropy |
|---|---|---|---|---|---|
| Recall | 88.255 | 80.956 | 81.168 | 83.431 | 99.202 |
| Precision | 98.506 | 99.367 | 79.431 | 94.071 | 97.105 |
| FPR | 3.773 | 2.170 | 4.131 | 1.547 | 1.576 |
| Accuracy | 89.077 | 83.530 | 69.515 | 65.370 | 96.407 |

the notion of similarity.

We considered botnets as threats worth of studying since cloud virtualization and service models may allow an easier path to their execution. Moreover, we aim to provide another experimental evaluation to the given found studies [84], [134] regarding this threat.

We utilized the CTU-13 Dataset [46], which comprehends real network traffic capture of more than 5000 hosts labeled in background, normal and botnet behaviors. In particular, this traffic concerns different types of DDoS, port scanning, C&C attacks, among others, there is no single threat pattern, and our experimental schema relied on the trial of arbitrary techniques against this range of attack patterns. Training and testing distributions were respectively 83.39% (50.57% and 49.43% for normal and botnet traffic) and 16.61% (13.95% and 86.04% for normal and botnet traffic), accounting more than 90 million packets. Results were analyzed by widely used metrics Precision, Recall, Accuracy and False Positive Rate (FPR).

Data preparation consisted in reading NetFlows[1], selecting and normalizing their attributes in *header-based features* (e.g., source IP address, destination IP address and port, protocol), *content-based features* (e.g., source bytes) and *time-based features* (e.g., session duration). As tried to simulate the monitoring of continuous data streaming flows arriving from the cloud, all techniques were implemented using *online learning*, by feeding the algorithms with timely ordered dataset in batches.

In particular, SVM was used as a binary classifier. We applied it with linear kernel, taking into account good experimental results presented in [71]. For MLP, experiments consisted of finding hyper-parameters values and analyzing their impact against the detection metrics mentioned. Given the low standard deviation while changing the number of training iterations, we proceeded experiments with this parameter fixed at 50 epochs. Model setup was a two-layer hidden network, with 36 hidden neurons each. The variability of the latter consisted in increasing the number of neurons, obtaining higher recall and precision values, but also raising the FPR.

For LSTM, we also experimented with various training parameters and topologies. Hidden layer consisted of two LSTM memory blocks, with two cells each and peephole connections. Adam algorithm [72] was considered as the optimizer while MSE as a loss function. We applied an arbitrary exponential learning decay of 0.97. Time step size, batch size and epoch in ranges from [10,200], [50,500], [50,800] respectively, while modifying the learning rate from 0.0001 to 0.1.

For K-means technique we applied the Mini Batches function, a faster approximate version of the more "expensive" K-means clustering [115]. The configuration was set for the algorithm to create two clusters, normal and abnormal (botnet traffic). This configuration falls upon the thesis's assumption, i.e., normal connections are frequent whereas attacks are very rare,

[1]Network protocol developed by Cisco for the collection and monitoring of network traffic flow data generated by NetFlow-enabled routers and switches.

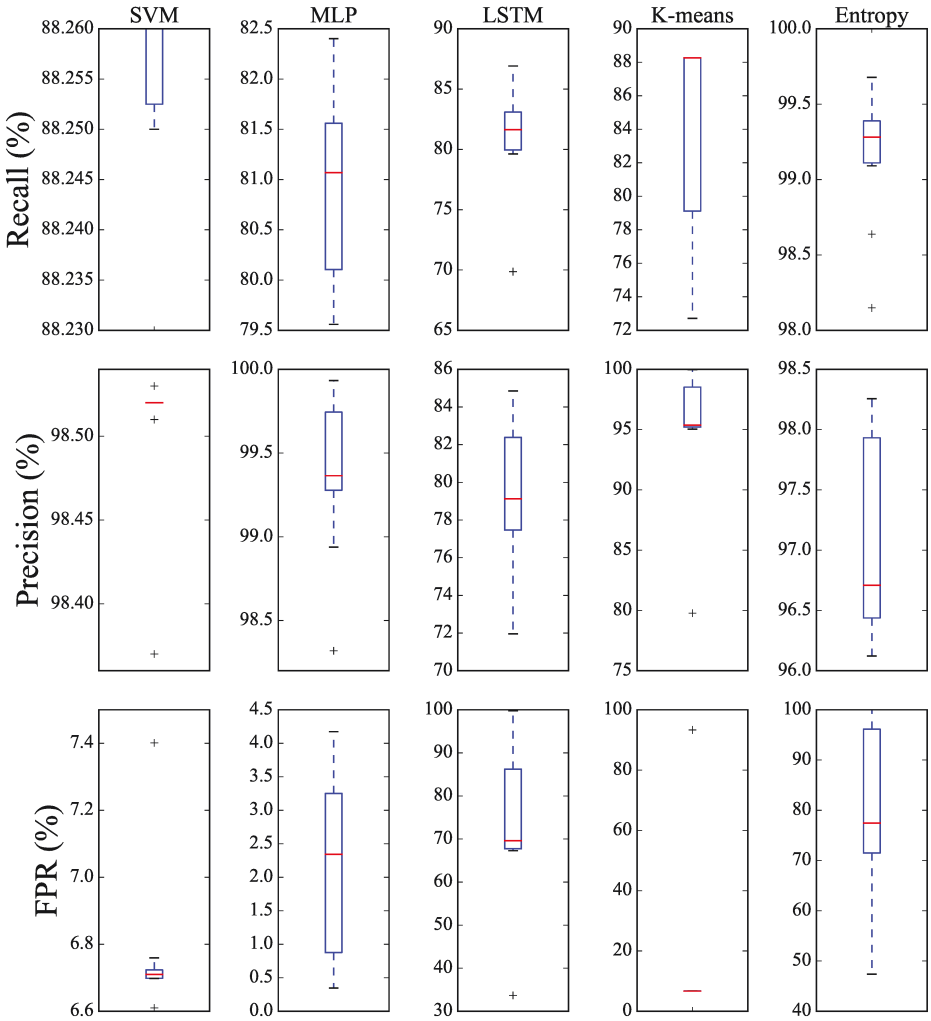



Figure 2.3 – Experimental results for SVM, MLP, K-means, LSTM and Entropy-based

the clustering algorithm should create large clusters for normal connections and small isolated clusters for anomalies.

Lastly, we used an entropy-based detection following an information theoretic perspective. We calculated the entropy $H(i,k)$ for each streaming batch $i$ of $k$ traffic features. Every network flow was treated as a point in a 6-dimensional space with coordinates

$$\langle \overline{H_i} \rangle = \langle \overline{H}(srcip), \overline{H}(srcbytes), \overline{H}(dstip), \overline{H}(dstport), \overline{H}(proto), \overline{H}(dura) \rangle$$

To analyze this multivariate vector in a simpler single-dimension representation, we re-scaled each sample $\langle \overline{H_i} \rangle$ to $\| \langle \overline{H_i} \rangle \|$ in ranges of [50,2000] samples per batch. Consequently, *normal* behavior is defined as the projection of the data onto this subspace and *abnormal* behavior is defined as any significant deviation of the data from this subspace. The static threshold is defined as the norm of all the calculated $\langle \overline{H_i} \rangle$ training entropies and tried against the testing dataset.

We summarize our experimental results in Table 2.1 and in Figure 2.3. The performance

of SVM showed to be more stable than the others for all the metrics, while Artificial Neural Network-based techniques depended on the topology and training parameters. One can see from Table 2.1 that overall the techniques perform above 79% for Recall and Precision indicators. Nevertheless, LSTM and Entropy techniques do it also by increasing the FPR. The latter is probably due to our static threshold configuration and that chosen features may have dismissed or not fully exploited the dataset threat behavior.

## 2.6 Insider threat related works

In virtue of the literature mentioned above review and analysis, together with the proposed grouping of detection techniques, we were able to see the different aspects of all threats and the approaches for solving them. Consequently, we have decided to tackle in depth the study of detection towards the insider threat, which is one of the most difficult studied threats, due to its challenges regarding monitoring, data accessibility and later processing for its detection, among others.

**Cloud and non-cloud related detection techniques** The insider threat issue has been studied from different optics in the current literature. From this fact, an overview of the past approaches can be found in [51], [82]. These works cover multiple detection approaches and use-cases. As an example of them, we can see various categories such as: real-time and non-real time, host-based, network-based monitoring levels, supervised, semi-supervised and unsupervised, as well as different detection algorithms such as machine learning-based, information theoretic-based, among others.

Statistical approaches assume that the data follow some standard or predetermined distributions, and this type of approaches aim to find the outliers which deviate from such distributions. However, most distribution models are assumed uni-variate, and thus the lack of robustness for multidimensional data is a concern. Nevertheless, the assumption or the prior knowledge of the data distribution is not easily determined for practical problems.

In distance-based methods, the distances between each data point of interest and its neighbors are calculated. If the result is above some predetermined threshold, the target instance will be considered as an outlier.

In density-based methods, the approach is to use the density (i.e., well connected points), such as local outlier factor (LOF), to measure the "outlier-ness" of each data instance. For example, based on the local density of each data instance, the LOF determines the degree of "outlier-ness", which provides suspicious ranking scores for all samples. However, it is worth noting that the estimation of local data density for each instance is very computationally expensive, especially when the size of the dataset is large.

Though not initially intended for detecting the insider threat in cloud environments, the previously mentioned techniques contribute to how to analyze users interacting with assets from many perspectives and utilizing them for the scope of this work, where the insider threat is examined by its actions in a cloud environment.

Machine learning approaches have also been used for tackling this threat. An example is Tuor et al. [127], which used real-time unsupervised Deep Neural Network (DNN) and Recurrent Neural Network (RNN). They treated their system in an online fashion by training a single RNN with a supplementary data structure that stores a finite window of past inputs and hidden and cell states for each user. Each time a new feature vector arrives into the model, the hidden

and cell states for that user are then used for context when calculating the forward pass and back-propagating error. Although their model benefits from the internal relationships of DNN, it begins outputting verdicts while completely untrained, therefore anomaly scores are very high for the first period.

In another schema, Yen et al. [141] presented a machine learning approach that analyses different network-based logs in an enterprise, namely proxy, Dynamic Host Configuration Protocol (DHCP), Virtual Private Network (VPN), Lightweight Directory Access Protocol (LDAP). They later utilized PCA with the K-means clustering algorithm with labeled data for evaluation purposes.

Also PCA was used by the authors in [77], which employed unsupervised feature transformation on their own developed dataset. Their system then creates a tree-structure for each profile, constructs set of features that describe particular anomalies of interest and computes a PCA decomposition to identify features that exhibit high deviation.

As an example of a distance-based approach, authors in [12] proposed an unsupervised k-Nearest Neighbors (k-NN) with a k-d tree data structure in order to find the k-NN of each new data input. The smallest distance (nearest neighbor) to this new observed point is treated as the anomaly score of the input. If the anomaly score exceeds a fixed threshold, then the point is treated as anomalous. This criterion relies on the common drawback of having high false positives using a fixed threshold while newer data inputs may vary and evolve through the employee's behavior in time.

Along with the presented approaches, state-based approaches can be seen for anomaly detection in time-series data. These models assume that there is some underlying hidden state and previous state that generate the observations and that this hidden state evolves through time, possibly as a function of the inputs. Rashid et al. [108] used Hidden Markov Model (HMM) which they trained for a five-week period. Later they were able to predict further inputs' anomalies against a fixed threshold.

Graph-based approaches have also been introduced into the insider threat detection. They intend to model user-system interactions through topological properties, e.g., system components and user nodes can be connected if they have interacted between each other. Parveen et al. [103] utilized using system-calls data and proposed a graph-based approach to create multiple models for a batch of system calls and their parameters which, finally, constitute an ensemble to detect subsequent batches in a streaming manner. During each iteration, a batch is tested with all the models, and a weighted majority voting mechanism is applied to make a decision. Afterwards, the least weighted model is replaced with the new model.

In [34], the authors have used Email and Cell phone logs, specifically analyzing the correspondence patterns. A normative pattern (a graph substructure) is learned from the entire graph that describes an insider's correspondences by minimizing the description length (MDL) and an incident of interest is raised when a test graph substructure is inconsistent with the normative pattern (by looking for modifications, insertions, and deletions in the graph structure).

In [91], the authors proposed a bipartite graph between two types of nodes: users and system components. Their methodology includes studying the evolution of the user graph to identify topological properties that characterize the system's normal behavior. Among these observed properties, those that do not follow the norm of the regular pattern are assumed to indicate the presence of an anomalous event.

Additionally, within the scope of graph-based approaches, tree or partitional techniques have also been used lately [45], [49]. Gavai et al. [49] utilized a modified version of the Isolation Forest (IF) method at the task of detecting insider threat from network logs. The process continues

until each individual point in the dataset is isolated in a leaf. Every point receives a score that relates to the average number of splits required over the IF trees to isolate it into a leaf node. They modify this algorithm to identify the anomalous points that arise from the averaging across the forest and also record the corresponding features that are the reason for their isolation.

Other authors [45] use IF with logs from LDAP, proxy and email services, and apply the algorithm to detect suspicious users by looking into the graphs and their sub-graphs.

Non-insider threat related anomaly detection studies also contribute to the knowledge in discovering anomalies as a data mining practice in general (an extensive survey can be found in [105]). Along with clustering proposals such as *ClusStream* under the seminal work of [3], there can also be found techniques within the previously named categories.

One of these sets of algorithms worth mentioning is SOM-based network models, which are suitable for online processing of unlabeled data in numerous anomaly detection applications [13] as much as in the security domain [47], [126].

**Cloud-based related datasets** Authors of [70] proposed a cloud-based dataset for masquerade attacks, i.e., where an attacker assumes the identity of an authorized user for malicious purposes. They utilized network and host traces from two machines of the DARPA dataset [81], consisting in host-based audits from Windows NT and Unix Solaris, along with their corresponding Transmission Control Protocol (TCP) data. They correlated a seven-week dataset and labeled the users from both machines into different roles according to their login session time and the characteristic of the user task (e.g., programmer, secretary, system administrator). Later they assigned every user to a labeled VM.

Additionally, non-cloud related literature on dataset generation shows a variety of approaches. RUU dataset was provided by [113], also concerning masquerade attacks. They built a sensor host for Windows OS that captured user's registry actions, process execution and window touches. They collected normal users and analyzed differences against masquerade users, following a controlled exercise.

Carnegie Mellon's Computer Emergency Response Team (CERT) generated a collection of synthetic insider threat test datasets [14] to produce a set of realistic models.

ISCX dataset was proposed by [120], under the notion of profiles that contained detailed descriptions of intrusions and abstract distribution models for applications, protocol and lower network level entities.

The ADFA dataset [128] was proposed by [30] with modern attack patterns and methodology. This dataset was composed of thousands of system call traces collected from a contemporary Linux local server, with six types of up-to-date cyber attacks involved.

## 2.7 Conclusion

In this Chapter, we studied the relation between security threats and detection techniques in cloud environments. As a result, we conclude that data-related threats and malicious insider activities are mostly pursued by rule-based detection techniques. On the other hand, network-based threats such as DoS and botnet attacks can be effectively tackled with statistical and machine learning techniques. Likewise, whenever behavior-based or hybrid approaches are used, training data phase remains crucial to establish a wide spectrum of *normal* behaviors in cloud architectures. In this sense, more research needs to be performed to correctly discriminate them from real threats.

Through Figure 2.2, we have granted a visually synthesized comprehension of which algorithms have been studied for specific threats. However, we noted the absence of some links between them, raising questions regarding the use of certain techniques for threat detection. We think this may be because "well-accepted" methods have proven to be effective to known attack patterns in the past (e.g., SVM and DoS). On the other hand, the existent links are due to the tryout of novel techniques against classic threat patterns or the use of traditional techniques on top of cloud-environment settings.

The latter motivated to study the applicability of existing detection approaches against new threats. Consequently, we attempted at experimenting with a SVM, two ANNs, a statistical and a clustering method; performed an online detection and obtained results to counteract these unseen techniques with a dataset of recent malware vectors. In particular, it was experimentally proven that SVM behaves well as an "all-around" classifier, keeping good accuracy while low false alarm rates. In contrast, we observe additional studies should be pursued for neural network detectors as they rely on more parameters. This characteristic adds more complexity at the moment of detecting different types of threats, as they are commonly tuned for a particular test-bed.

At the same time, we note that although many of detection techniques have evaluated their accuracy given FPR, Precision; only a few studies are testing their performance in a holistic approach that contemplates specific cloud computing characteristics (named in Section 2.3), such as scalability, fault-tolerance or adaptiveness.

Additionally, high throughput interfaces and maintainable knowledge database repositories demand a scalable solution. At the same time, cloud-dynamic behavior varies regarding CSC's needs, and it can imply the discovery functionality for modified IaaS, PaaS or SaaS configurations. Therefore, it is important to keep in mind a flexible implementation approach that can detect anomalies adapted to each new requirement.

Furthermore, we have decided to tackle in depth the study of detection techniques towards one of the most difficult enumerated threats, such as the insider threat. By being one of the least treated threats due to its monitoring and detection difficulties, the assessment of a detection framework for this threat has also gained momentum in the last years in the literature. Nevertheless, many of the presented approaches solve the detection of anomalous behaviors only partially in a cloud-related scenario.

Each category of methods discussed in this paper has its own strengths and weaknesses, and faces different challenges for complex datasets. We consider some of the studied methods presented in [105], such as SOM-based methods, which benefit from the ability to address high-dimensionality problems, and the flexibility of having no *a-priori* assumptions about the properties of the data distribution, therefore allowing the adaptation of the learning process in time.

Along with the discussed threats and techniques, this state of the art approached in depth towards the insider threat. With regards to this threat, the literature mentioned above raises many questions about the proper characterization of malicious insider threat and which features could adequately describe it for later detection techniques' analysis. Additionally, most of the presented datasets correspond to one-time implementations, which limits the generation and analysis to that particular test-bed configuration. In this respect, an automatic dataset generation is aimed, which will establish different scenarios with more dynamics taken into consideration. This feature also makes the analysis modifiable, extensible and reproducible.

Finally, this survey also proves the absence of a universal approach for identifying various threats of different nature. Additionally, focusing on multiple cloud service features will provide

an integral perspective of different behaviors working together. Developing such approach or at least making steps towards deriving a broader yet effective cloud threat detection system, without a doubt, form a group of hot topics for future research work.

Chapter 3

# Modeling the Insider Threat

**Contents**



## 3.1 Introduction

In this Chapter, a model for the insider threat is presented. This model is an abstraction of a user or employee in a company executing actions towards a cloud environment. The main objective is to describe the most relevant aspects to consider at the moment modeling a user and/or malicious insider, by means of utilizing psychological and technical attributes.

In regard to this threat, the previous Chapter 2 in Section 2.6 considered different detection approaches, which either utilized collected data from different logs, or obtained datasets by means of simulating users interacting with different resources. The first group presents some shortcomings regarding experimental reproducibility and only solves the detection for that specific dataset. The second group, although it presents different dataset generation approaches, new enhancements should be made in order to address the threat from a cloud-environment perspective.

Moreover, few insider threat models and datasets have been proposed in the literature, and much fewer indicators related to cloud-based insiders. Thus, the proposed model considers the aforementioned discussion made in Section 2.6 regarding existing cloud-related datasets, by proposing a richer behavior simulation, which is reproducible and versatile. The former enrichment makes use of psychological factors given in the literature and proposes an associated cyber-factor.

The structure of the chapter begins with the preliminaries (Section 3.2) and the main concepts related to the insider threat, such as its definition and main characteristics. Later in Section 3.4, the user model is presented, concerning the insider attributes and how this model generates data for posterior detection goals. Section 3.5 gives detailed insights on the malicious behavior for the user model, while Section 3.6 presents an evaluation for the user model modules, and also describes a validation criteria, while the last part of the chapter (Section 4.5) ends with the conclusions.

## 3.2 Preliminaries

**Insider threat taxonomy** This threat has been modeled in numerous studies. An ontology-based definition was provided by [29] using the real-world case data, considering factors such as (i) Human behavior; (ii) Social interactions and interpersonal relationships; (iii) Organizations and organizational environments; and (iv) Information technology security. Another taxonomy was considered by [65], where a malicious insider is composed of (i) System role (novice, advanced, administrator); (ii) Sophistication (low, medium, high); (iii) Predisposition (low, medium, high); (iv) Stress level (low, medium, high).

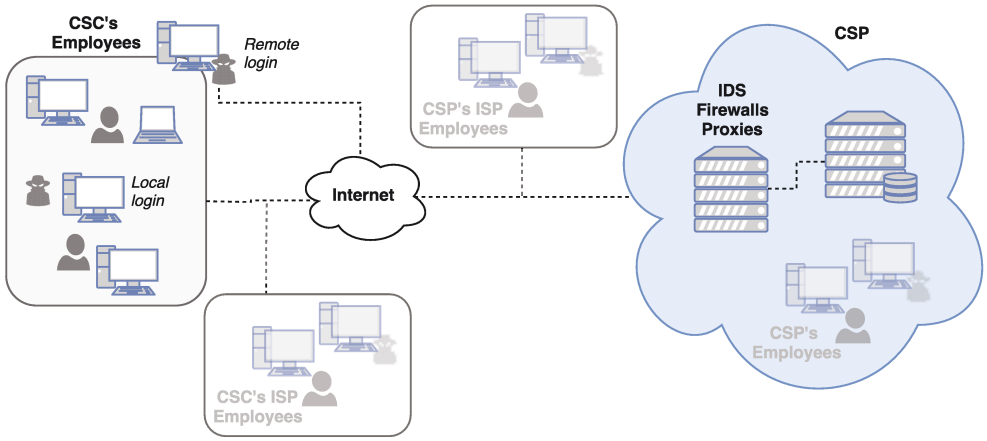



Figure 3.1 – Malicious Insider in the cloud from end-user perspective

However from Figure 3.1 one can see that this threat in cloud-environment increases in

complexity taking into account new actors involved, as well as their dependencies. Consequently, a role-based categorization was proposed by [26], where an insider threat could be (i) A malicious insider from the CSC, accessing cloud services or (ii) A malicious insider from the CSP, accessing to sensitive company data. In addition to these, more actors were presented by [33]: (i) A malicious insider from the Internet Service Provider (ISP) for each zones; (ii) External CSPs if resources are outsourced to other providers; and (iii) Cloud provisioning services (brokers).

User profiling research and recollection of real cases [28] present no common pattern with respect to subject's psychological characteristics. However there are risk indicators [53], [78], [123] related to the motivational factors that may underlie malicious insider exploits, which are supported by studies indicating that most of these attacks (81%) are planned [119].

From these representations of insider user profiling, we derive our model definition in the following section. Moreover, we propose a threat ontology with a probabilistic approach, where the disposition factor to perform malicious activities mentioned above is considered to occur with a given probability in time.

**Insider's opportunity** A generic set of models usually refer as the Capabilities Motives and Opportunity (CMO) model [102], [112], [114]. This model postulates that to commit an attack, the insider must first have the: 1. Capability to commit the attack, 2. Motive to do so, 3. Opportunity to commit the attack.

These three attributes are also present in our model, and in addition, we illustrate how an insider may relate each of these and come to perform the malicious act through the relationship with other elements. In particular, the notion of the opportunity dimension is well defined within the literature relating to threat assessment and risk management [97]. Given the capability (e.g., skills) and motive (e.g., revenge, greed), the insider must also have an opportunity. This opportunity is enhanced when the insider is able to exploit a weakness present in the organization's assets.

As mentioned, while this attribute has been well defined within the rational decision literature with respect to threat assessment and risk management [62], [97], there have also been models proposed regarding insiders' perception of risk and endogenous characteristics that are unique to insiders [40], [53], [123]. The two main key notions are that (i) in order to perform a malicious act, an employee has to perceive an opportunity; (ii) when this opportunity is perceived, the employee also has a perceived risk in terms of the action's potential consequences (i.e., getting caught). Therefore, while traditional risk assessments focus mostly on technical vulnerabilities, insider threat assessments need two more dimensions: the employees' and organization factors (e.g., policies, practices).

In addition to that, in the context of a cloud environment the relationship between the employees and the organization's assets, lead to more complex vulnerabilities. Moreover, as relevant cloud-based characteristics were addressed in Section 2.2.1, there are several security challenges inherent to this environment.

An insider threat assessment is a statement of threats posed by 'trusted' insiders of an organization that are related to vulnerabilities, assets, and insider threat agents. In order for a malicious insider to bring its capability to bear against a target, the employee must have the correct conditions to do so; and in order for their capabilities to be effective and have an impact on the target, the target must be vulnerable to attack.

Consequently, we define the levels of opportunity for an insider and follow quantification methodology of authors in [112], namely: 1. Define the vulnerabilities associated to the assets interacting with him or her; 2. Highlight the security controls that are appropriate to mitigate

the exploit of the asset. Concerning the first aspect, ISO 27005 [96] defines a vulnerability as:

> *A weakness of an asset or group of assets that can be exploited by one or more threats.*

We extend this description with European Union Agency for Network and Information Security (ENISA)'s more specific annotation:

> *The existence of a weakness, design, or implementation error that can lead to an unexpected, undesirable event compromising the security of the computer system, network, application, or protocol involved.*

By means of assessing the different vulnerabilities, the CSA provided with recommended controls for all the previous detailed threats from Section 2.2.2. They released the "Security Guidance for Critical Areas in Cloud Computing" and the "Security as a Service Implementation Guidance", providing best practices for securing Cloud Computing (CC) infrastructures. Some of the recommended security controls for the malicious insider threat are

- Compliance (CCM CO-03): Third-party audits.
- Data Governance (CCM DG-01): Information leakage.
- Information security (CCM IS-08): User access restriction/authorization.
- Human Resources Security (CCM HR-01): Background screening.
- Information Security (CCM IS-19): Audit tools access.
- Information Security (CCM IS:18): Encryption.

## 3.3 Proposed modeling approach

The definition of an ontology involves complex interconnections in different domains. The following areas have been considered, in order to better characterize employee's activities in an organization.

**An insider is modeled by two main attributes**, namely Intra-psychological behavior and cyber behavior:

- **Intra-psychological behavior** considers the humans' state of mind behind the malicious activity i.e., the reasons why they performed the malevolent actions, their psychological motives and the characterization of their intent.
- **Cyber behavior** considers humans' behaviors in relation with a cyber cloud asset i.e, the way employees interact with the organization's resources.

Additionally, we itemize our assumptions regarding the behavior of an insider with respect to the rest of the organizational cloud-related assets, along with the dataset generation.

**The insider threat is a rare event** hence, we consider its occurrence rather low. Therefore, as treated as an anomaly, we consider the probability of an opportunity is low in most of the organizational environments.

## 3.4 User Model

The model proposed in the following paragraphs aims to consider each one of these domains as part of the abstraction of an employee, and moreover interrelate them to derive more complex scenarios for posterior detection analysis.

### 3.4.1 Definition of entities

We consider the following entities to generate patterns of activity, as shown in Figure 3.2. They are divided into two groups. On one hand user-related entities, namely:

**Profile** is defined as an abstract representation of person's attributes in an organization, to facilitate the reproduction of realistic behaviors. Each `Profile` is composed of a `Psychological factor`, a `Cyber factor`, a `Context` and a `Role`.

**Psychological factor** is related to the human characteristics of a person. This category adds a dynamic realism by means of human attributes, such as the attitude for the given job. This factor can be personalized and distinguished by its intention in time.

**Cyber factor** is related to the professional or technical characteristics of a person based on their `Role`. This category adds attributes such as the way of interacting with the company's resources.

**Role** is associated to the job of the `Profile` in a given organization. Moreover, the `Role` is defined through the entity `Policy`, which is composed of a `Permission` related to an action towards an `Asset`.

**Context** consists of attributes related to specific time and location conditions where the `Profile` is performing its `Role` (e.g., location from where the actions are being executed, time of the day, IP from where actions are being executed, cloud instance the employee is trying to access).

**Permission** is the type of authorization a `Role` has for a given `Asset` (e.g., read, modify).

**Asset** consists of any valuable hardware or software component, property of a CSC in the CSP stack (e.g., physical servers, VMs, applications, databases, communication infrastructure), depending on SaaS, PaaS or IaaS models.

On the other hand, the simulation also includes the event-related entities:

**Sequence** is a list of (sequential) actions performed by the same `Profile` under a given time interval. For that matter, an *action* $a$ is defined as a symbol in an alphabet $\Sigma$. Based on this alphabet, the approach generates three types of `Sequences`:

(i) Pseudo-random sequences, defined as a Set $R \subset \Sigma^*$ of finite pseudo-random actions.

(ii) Predefined sequences, defined as a Set $P \subset \Sigma^*$, where each of its strings' letters is assembled together following a lexicographical order, by means of *normal* and *anomalous* actions, under realistic scenarios.

(iii) Hybrid sequences, composed by at least two fixed actions in arbitrary positions, filled with pseudo random actions for the rest of the sequence. Thus, an example of a *hybrid sequence* has the form $(p_0, r_0, \ldots, r_n, p_n)$ where $p_0$ and $p_n$ are predefined sequences.

**Label** represents the nature of the anomaly occurred (e.g., *normal*, *anomalous*).

**Event** consists in the tuple of attributes

$$\langle\, timestamp, profile, sequence, context, label \,\rangle \tag{3.1}$$

Which corresponds to the observation of a *sequence* with a specific *label*, performed by a *profile* in a *context* for a given simulation time. These events can be generated with a given distribution in time (e.g., Uniform, Normal, Poisson), depending on the scenario of study, i.e., a Normal distribution could model rush-hours (e.g., end-users accessing a website application).

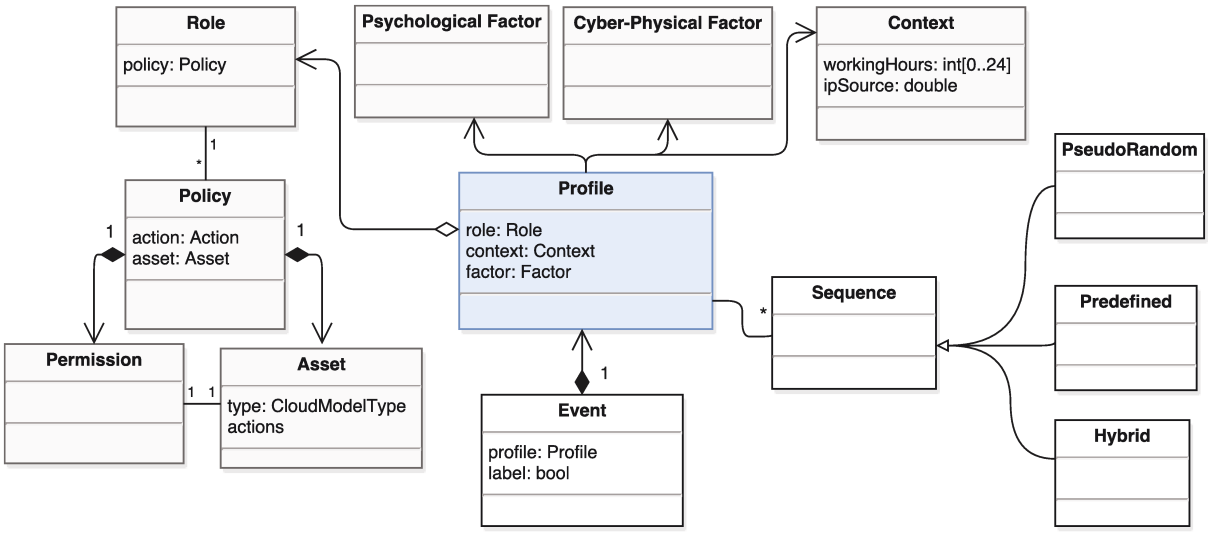



Figure 3.2 – Class models for generation of cloud insider threat dataset

### 3.4.2 Dataset generation

The Algorithm 1 presents the Agent-Based Modeling (ABM) dataset generation methodology. This process may be initiated by a list of `Profiles` (where each one is an agent), the inter-arrival time for every agent (*tba*) and the general simulation timeout (*timeout*). There is also a disposition threshold related to the psychological factor, and lastly weights for the different anomaly types. The *tba* parameter has relation with the amount of `Agents` to generate, and therefore with the `Event` generation process. Below, we provide more details related to the proposed algorithm referring to the corresponding lines of it.

Lines 2 to 7: Algorithm defines the `delay` for which it will create an `Event` from a pseudo-randomly chosen `Profile`. This generation is done with an exponential distribution, in order to model the pseudo-random generated sequences as a Poisson process (line 6).

Lines 10 to 14: An `agent` function takes the form of a `Profile` assigned by a `Role` and Time Between Events (*tbe*) with a given distribution (e.g., For Normal distribution with *mu* and *sigma* given by the `Context`).

Lines 17 to 18: The three types of `sequences` will be generated. Each of these group of sequences is given by the `Policy` entity, which relates an action to a given cloud `Asset`. The algorithm allows to give specific weights ($pol.seq\_weights$) to the three types of `sequences`. Later, the event is executed for that particular assignation.

Lines 20 to 23: Under a certain disposition given by the `Psychological factor`, each `Profile` will have an *anomalous* behavior. The function `GenAnomaly` can either change the `Context` or the `sequence`, introducing a single instance of such anomaly.

Line 25: The `agent` *sleeps* until a next scheduled event, which will be again *normal* or *abnormal* of a certain type. This is handled by the mentioned *tbe*, which takes values under the "working hours" `context` from the `Profile`.

**Algorithm 1** Dataset generator

1: **function** EVENT_GENERATOR($profiles, tba, timeout, disposition\_thresh, w_{loc}, w_{wh}, w_{seq}$)
2: **while** time $<$ timeout **do**
3: $profile \leftarrow pseudo\_random\_choice(profiles)$
4: $delay \leftarrow exponential\_distrib(1/tba)$
5: AGENT($profile$)
6: WAIT($delay$)
7: **end while**
8: **end function**
9: **function** AGENT($profile$)
10: $role \leftarrow profile.role$
11: $ctx \leftarrow profile.context$
12: $tbe \leftarrow normal\_distribution(ctx)$
13: $disposition \leftarrow$ PSYCHOLOGICAL_FACTOR($profile.psycho\_attrs, disposition\_thresh$)
14: $label \leftarrow$ "*normal*"
15: **for all** $pol \in role.policies$ **do**
16: **if** *disposition* **then**
17: $seq \leftarrow$ CHOOSE_SEQ($pol.rnd\_seq, pol.predef\_seq, pol.hyb\_seq, pol.seq\_weights$)
18: RUN_EVENT($seq$, *probile*, *label*)
19: **else**
20: $anomaly \leftarrow$ GEN_ANOMALY($profile, w_{loc}, w_{wh}, w_{seq}$)
21: $label \leftarrow anomaly.label$
22: $seq \leftarrow anomaly.seq$
23: RUN_EVENT($seq, profile, label$)
24: **end if**
25: WAIT($tbe$)
26: **end for**
27: **end function**

### 3.4.3 Psychological factor

**Algorithm 2** Psycho-social factor for agents in simulation

```
Obtain AFFECTIVE_COMPONENT D_t
if ΔD > τ then
    a_t ← RATIONAL_COMPONENT
    return a_t
else
    return 0
end if
```

As above-mentioned, this attribute aims at modeling the employee's psychological disposition with respect to performing a malicious act. Furthermore, each `agent` has the potential to become a malicious insider based on a combination of emotional and rational affecting their affective disposition and decision process, correspondingly. For the representation of this `Profile`'s attribute we utilized Epstein's [37] theoretical *Agent_Zero*, which is an ABM endowed with distinct emotional/affective, cognitive/deliberative, and social modules. This structure is grounded in contemporary neuroscience, where internal components interact to generate observed, often far-from-rational, individual behavior.

Unlike [37], who treats the three mentioned components (each belongs to the interval $[0, 1]$) as additive parts of the equation to be compared against a general threshold, we propose that an `agent`'s disposition to perform a malicious activity, is going to be based primarily by the affective component. This takes place under the hypothesis that an emotionally satisfied employee would not perform an illicit activity even if the rational component may be appealing (e.g., if there was a low risk and/or high reward associated). Following that idea, only if the affective component's value exceeds its personal disgruntlement threshold, the `agent` will secondly perform a cognitive decision process considering personal judgments (e.g., the value given to a reward), along with organizational context variables. This decision process will determine the optimal action to be taken. The above-mentioned general methodology for obtaining the psychosocial factor is formalized in the Algorithm 2.

#### 3.4.3.1 Affective component

To represent this element we use Epstein's affective component of the *Agent_Zero* model. In it, he uses the seminal Rescorla-Wagner model of conditioning [109], which is based in Reinforcement Learning (RL) and depicted as equation 3.2

$$\Delta D = \alpha(F_t - E_t) \tag{3.2}$$

This equation is also used by Sokoloswky et al. [123] in their specific model for the insider threat. They represent $\alpha$ as the level of attention the `agent` gives to the disgruntlement $\Delta D$ (also known as salience). This value can vary between 0 and 1, where 0 indicates that the affective component is irrelevant and 1 indicates that it has maximum relevance.

$F_t$ represents the observed outcome (commonly known as *reward* or *punishment*). Following [123]'s lead, it is the actual level of fulfillment the employee interprets from the organization. Lastly, $E_t$ will be the `agent`'s personal *expectation* of fulfillment.

In other words, for each simulation time $t$, the difference between the expected fulfillment and the actual experienced fulfillment will determine the level of surprise or *disgruntlement* $\Delta D$ towards the organization. This is interpreted as the employee's ability to observe, learn or adapt himself in regarding the organization environmental characteristics (e.g., organizational culture,

internal policies). Following [123]'s definition, the *expectation* of a user at simulation time $t+1$ is

$$E_{t+1} = \frac{\omega_{init} F_0 + \omega_{actual} F_t + \omega_{historic} \overline{F}}{\omega_{init} + \omega_{actual} + \omega_{historic}} \tag{3.3}$$

Where $E_{t+1}$ is the expectation for the next time of the simulation, $F_{t=0}$ is the initial fulfillment (e.g., assumed to value 1 when the employee begins to work in an organization), $F_t$ is the actual level of fulfillment at simulation time $t$ and $\overline{F}$ is the average of fulfillment values throughout the simulation. According to each `agent`'s arbitrary configuration, weights (namely $\omega_{init}$, $\omega_{actual}$ and $\omega_{historic}$) can have different values, emulating different employees' characteristics, which for instance, may give more relevance to the overall historic fulfillment or their initial fulfillment, rather than the actual one.

In our model, we propose a definition for the fulfillment observed by the `agent` at time $t$ as

$$F_t = \begin{cases} 1 & \text{with probability } p_t \\ 0 & \text{with probability } 1 - p_t \end{cases} \tag{3.4}$$

Where $p_t \approx 1$ considering that in general, organizations *normally* have good environmental characteristics and that events that may negatively impact employees are rather scarce. Finally, if the `agent`'s disposition exceeds its personal disposition threshold $\tau$, then rational component will take place, as

$$Disgruntled \iff \Delta D > \tau \tag{3.5}$$

Figure 3.3 depicts two examples for the variables in Equation 3.2. We observe both agents start the simulation without any disgruntlement ($D_0 = 0$). As time goes by, the fulfillment experienced within the organization decays ($F = 0$, shown as the a dark blue line event at the top of the figure), while the expectation remained high ($E > 0$). In this case, there is an increment in the level of disgruntlement for both `agents`. This level begins to decrement as the posterior observed fulfillment values remain high ($F = 1$, shown as the light blue line event at the top of the Figure). The different disgruntlement decays for both `agents` may also be observable, along with the increase whenever a single unexpected event occurs. These different behaviors derive from the salience factor $\alpha$, from Equation 3.2, and from the mentioned weights in Equation 3.3.

#### 3.4.3.2 Rational component

The proposed rational component is based on the work of [64], also proposed in [123] and [40]. It is referred as a descriptive model of human decision making under risk. In our case, the model describes how an employee aughts to take actions that maximize their expected reward also considering their assessed or perceived opportunity, in relation with to the organization's cloud environment.

We propose a simplified metric that sufficiently models the employee's decision process, by considering the *perceived opportunity* with respect to their skills to exploit the organizations' vulnerabilities. Specifically, we consider the `agent`'s decision to perform a malicious act, is based on this parameter, as well as on their reward and risk values.

This environment is modeled considering organizational attributes, which are not entirely known by the `agent`. As mentioned in Section 3.2, all possible contextual variables (e.g., security policies, assets architecture, organizational culture) ultimately fall under a probability of

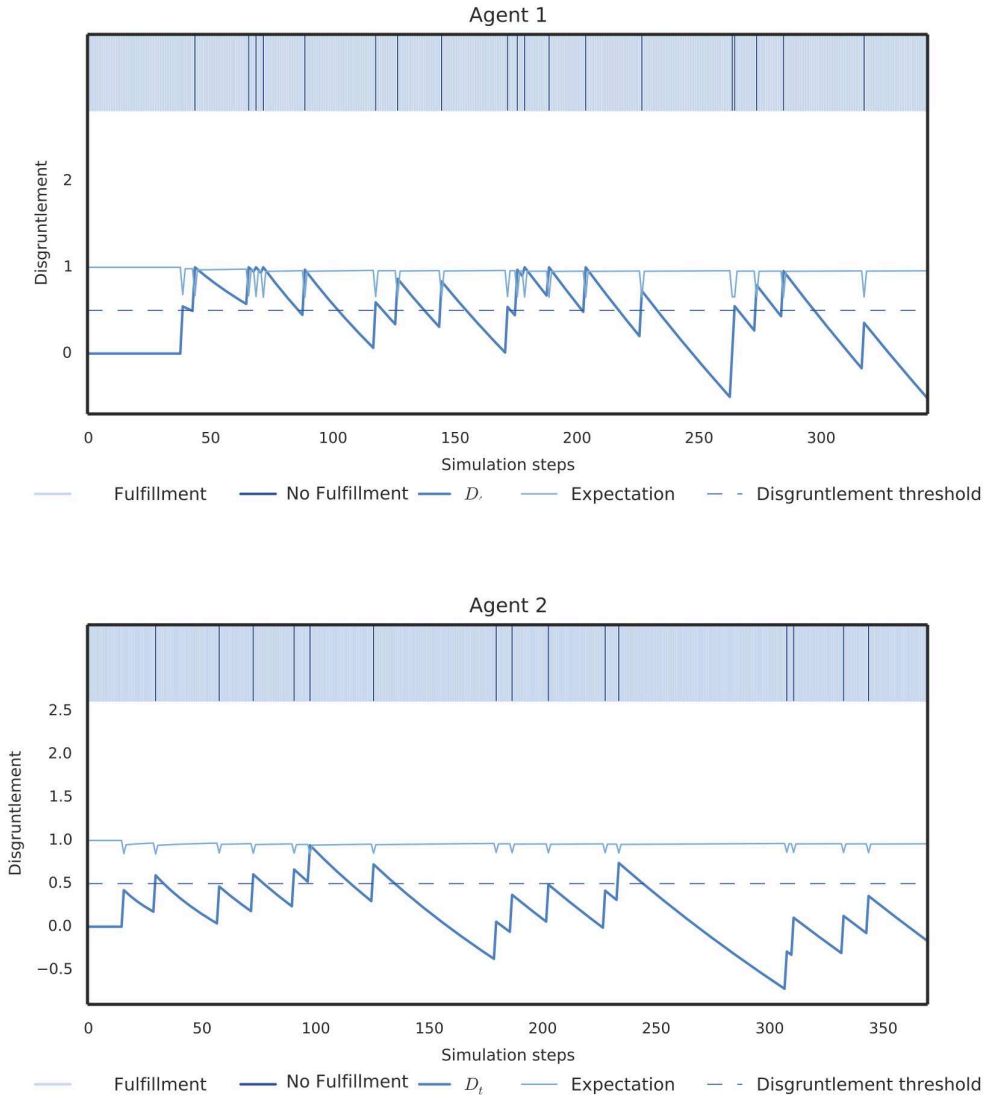



Figure 3.3 – Example of affective component for two simulated agents within a month period. Both agents have an disgruntlement threshold of 0.5, while the $\alpha$ is 0.8 and 0.5 respectively. $\omega_{init}$ is 0.001, while both give different relevance to the actual fulfillment value ($\omega_{actual}$ is 0.299 and 0.099, respectively). The assigned value to the historic fulfillment $\omega_{historic}$ is 0.7 and 0.9 respectively.

*perceived opportunity* to commit a malicious act. The quantification of opportunity *po_value*, is perceived subjectively as an arbitrary parameter by the `agent` and is based on the assessment of the system's vulnerability and the existence of implemented security controls.

Following this path, a methodology for deriving the opportunity perceived by the insider is proposed. The user reasoning considers four steps:

1. Identification of the organization's assets involved in the cloud implementation.

2. Identification of the Security Control (SC) that protects these assets.

3. Identification of which SCs are "implemented" or "not implemented" in the cloud environment.

4. Quantification of the number of not implemented SCs over the total of SCs, seen as the vulnerability level for each asset ($not\_pct\_implemented\_SCs$).

5. Estimation of the perceived opportunity of the insider.

Therefore, we define the perceived opportunity of an insider, aiming to perform a malicious activity towards an asset as

$$po\_value = \beta \cdot not\_pct\_implemented\_SCs \tag{3.6}$$

where $\beta$ is a parameter that can have three values, namely high medium and low, which are proportional to the skill of the employee (also defined as high, medium and low).

As mentioned in Section 3.3, we define it this way due to the fact that employees may not know all the organizational vulnerabilities associated when planning a malicious activity, but it could be partially estimated given the employee's experience or knowledge within the organization. Nevertheless, we consider this probability rather low, under the assumption that the insider threat is a rare event, therefore the probability of an opportunity is low in most of the contextual environments. In detail, the model is formed by the tuple

**Algorithm 3** Rational component of the psychological factor for agents in simulation

**function** RATIONAL_COMPONENT($t$, $reward\_value$, $risk\_tol$)
  Obtain a $po\_value$ from a given asset
  Obtain reward payoff value $V_{reward}$
  Obtain risk payoff value $V_{risk}$
  **if** $V_{reward} > V_{risk}$ **then**
    **return** $a_t \leftarrow$ Perform malicious activity
  **else**
    **return** $a_t \leftarrow$ Perform normal activity
  **end if**
**end function**

$$\langle\, po\_value, A, reward\_value, risk\_tolerance \,\rangle \tag{3.7}$$

Where $po\_value$ is the above-mentioned opportunity the `agent` perceives for pursuing a malicious activity (thus, the probability of risk is $1 - po\_value$). $A = \{perform\ malicious\ activity, not\ perform\ malicious\ activity\}$ is the set of possible actions to perform. The parameter $reward\_value$ depicts the relevance that the employee gives to the reward by means of the action *perform malicious activity*, while $risk\_tolerance$ is the degree of uncertainty of being caught that the employee is willing to withstand. Moreover, three levels of $risk\_tolerance$ and *reward value* have been defined. This values represent profiles such as risk-averse, risk-neutral and risk-greedy. For the *reward value* parameter sweeps, we have defined high, medium and low attractiveness.

Thus, at every simulation time $t$, the organization's environmental context is at an opportunity state with probability $po\_value$, according to the `agent`. Given this probability, the `agent` will choose an action $a_t \in A$ according to their personal decision policy. The above-mentioned general methodology for obtaining the rational factor is formalized in the Algorithm 3.

The policy used for this component is given by the relation between expected payoff values between the risk and the reward

$$V_{reward} = reward\_value \cdot po\_value \tag{3.8}$$

$$V_{risk} = \frac{1}{risk\_tolerance}(1 - po\_value) \tag{3.9}$$

$$Perform\ malicious\ activity \iff V_{reward} > V_{risk} \tag{3.10}$$

where $1/risk\_tolerance$ determines the risk-profile (i.e., greedy, neutral or averse). To illustrate this methodology, the following example is provided, followed by the Figure 3.4

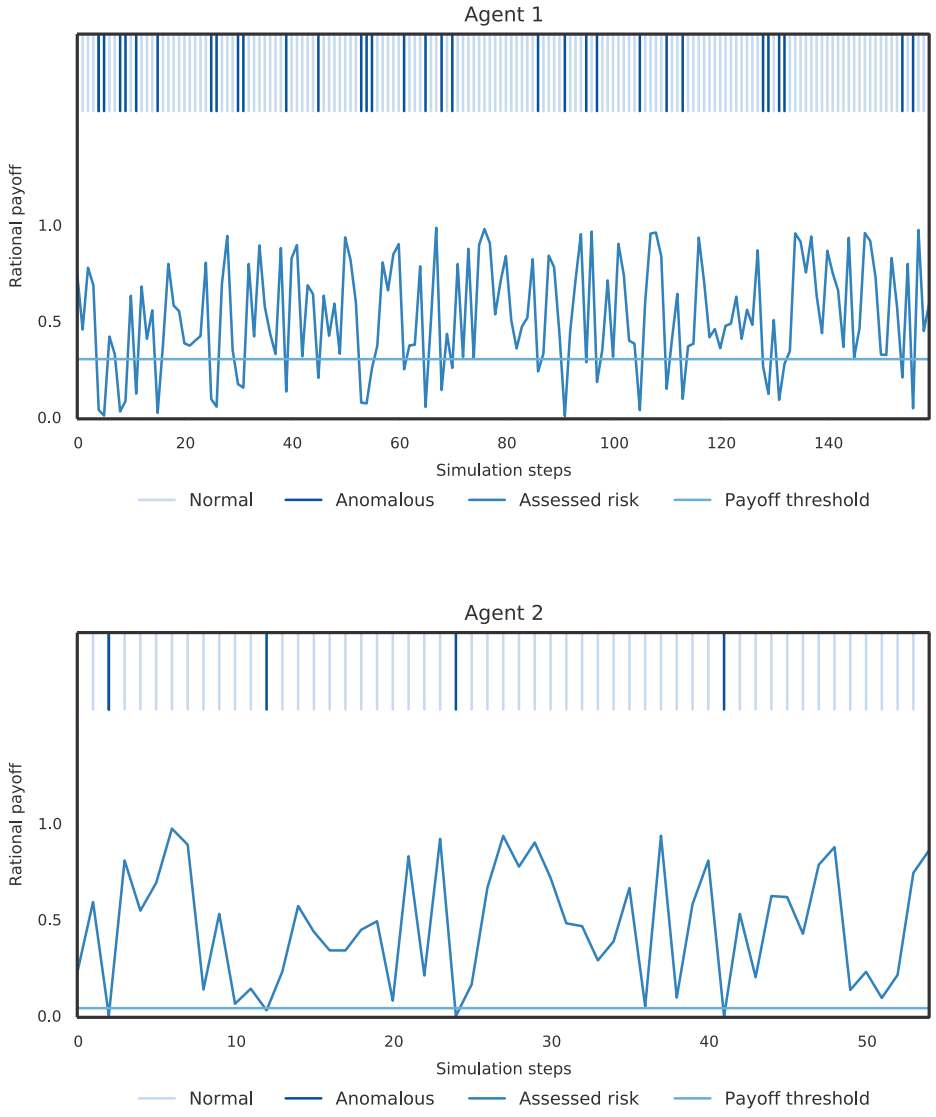



Figure 3.4 – Example of rational component for two simulated agents within a month period

*Example.* We define two `agent`'s (namely Agent 1 and 2) with different rational attributes. Both have same value to reward (0.5), nevertheless Agent 1 is risk-greedy (0.9) while Agent 2 is

risk-averse (0.1). For a perceived opportunity $po\_value \in U(0,1)$ the procedure for every step of the simulation is to compute the final decision through Equation 3.10. From the Figure 3.4 we may observe that Agent 1 (greedy) is more willing to pursue a malicious activity than averse Agent 2 (greedy).

Finally, if the action is to perform the malicious act, then an anomalous `Event` will be scheduled, as depicted in Figure 3.5.

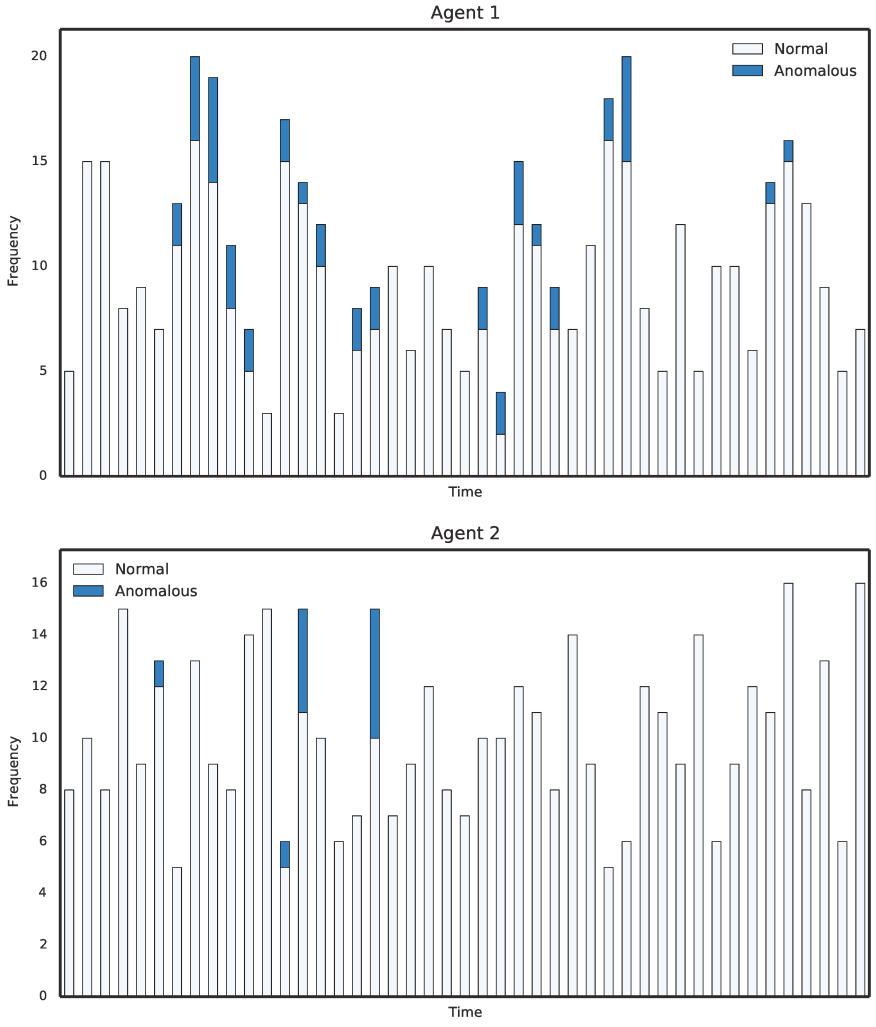



Figure 3.5 – Example of anomalous events for two simulated agents within a month period

### 3.4.4 Cyber factor

As previously mentioned, an `agent` has a specific `Role` that determines the relation with `assets` of their organization. These activities are modeled by the set of `Sequences` described in Section 3.4.1 and are generated through a methodology that has two major benefits. The first is the fact that it can help distinguishing some insider threats regarding data leakage or tampering, by detecting anomalies in the relation between their `Sequences` of activities and historic

frequency. The second advantage is the fact that it can help distinguishing an insider threat towards masquerade attacks, by means of analyzing the content of the `Sequences`.

The actions may vary according to the different `assets` and the level of granularity intended to simulate. For instance, as Figure 3.6 depicts, actions may differ for an `agent` whose role is a Developer, a DBA or a Cloud Administrator (formerly System Administrator) (CloudAdmin).

This set of actions will be also dependent on the cloud context regarding the architecture's configuration, such as service model, type of access layer for accessing the cloud services, execution commands for configuring the architecture, among others. Thus, for a Database (DB) asset a Developer could have actions such as `read`, while the DBA could have the former, in addition to `create_role`, `create_user` in the database user's access, or `create`, `update` and `delete` for the database records. The CloudAdmin on the other hand, interacts with the asset VM and its Operative System (OS), in case of an IaaS model. This interaction is done via the Command-line Interface (CLI) or an Application Programming Interface (API), which determines another set of possible actions.

As mentioned in previous Section 3.4.1, the cyber factor is based in these `Role-Asset` relations, where every `Role` has a predefined set of authorized actions. These actions are the base for deriving the *Predefined*, *Random* and *Hybrid* `Sequences`.

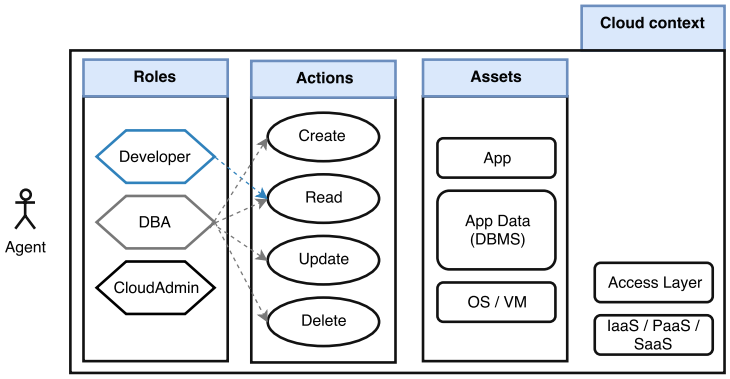



Figure 3.6 – Example of actions for a DBA and Developer with respect to cloud and organization assets

By means of generating different profile `agents` with the same `Role`, a cyber factor *Skill* is proposed and defined as the average time in seconds an employee takes to execute a set of actions. This attribute may be considered as a particular technical characteristic of the employee, as it relates with professional competences that allow the execution of actions in a more or less efficient, cautious or creative way.

As the formerly presented Algorithm 1 for generating the simulation data, for every time step $t$ a `Sequence` is obtained and an `Event` is scheduled. At the end of the simulation the result is a collection of `Events` $E$ for each profile.

Accordingly, let $E = [e_1, e_2, \ldots, e_n]$ be the collection of $n$ events where $e_n$ has the form of the tuple from Equation 3.1. Formally, for a single profile we initially have

$$E = \begin{bmatrix} t_1 & sequence_1 & context_1 & label_1 \\ \vdots & \vdots & \vdots & \\ t_n & sequence_n & context_1 & label_n \end{bmatrix}$$

The objective is now to group this `Event` matrix $E$ into sessions $S$ with respect to the *Skill*

**Algorithm 4** Methodology for grouping events into sessions

1: **function** CYBER_FACTOR(profile, malicious_labels, normal_label)
2: $E \leftarrow$ profile.events, where $e_i =< t_i, sequence_i, context_i, label_i >$
3: $D = \{\, d_i \mid d_i = t_{i+1} - t_i \,\}$
4: $I = \{\, i_i \mid d_i > skill\_threshold \,\}$
5: $StartEnd = \{\, (s_i, e_i) \mid (s_i, e_i) = (i_i, i_{i+1}) \in I \,\}$
6: **for all** $(s, e) \in StartEnd$ **do**
7: $session\_cnt \leftarrow i_e - i_s$
8: $duration \leftarrow t_e - t_s$
9: $session\_seq \leftarrow concatenate([sequence_s, sequence_e])$
10: **if** $[label_s, label_e] \in malicious\_labels$ **then**
11: **return** $label \leftarrow mode([label_s, label_e])$
12: **else**
13: **return** $label \leftarrow$ normal_label
14: **end if**
15: $context \leftarrow mode([context_s, context_e])$
16: **end for**
17: **return** $S = \{\{\, t_j, session\_seq_j, context_j, session\_cnt_j, duration_j \,\} \mid j \in StartEnd \,\}$
18: **end function**

parameter. The criteria for grouping the different attributes of the matrix $E$ into sessions are formalized in the Algorithm 4 and described in the following paragraphs.

As a first step, the algorithm calculates the time differences between all the events and collects it in set $D$ (line 3).

Secondly, from $E$ it obtains the matrix indexes where duration exceeds the *Skill* time i.e., threshold, and collects them in the set Index $I$ (line 4). In other words, consecutive events which time is lower than this threshold, will be considered as part of the same session $S$. Consequently, it builds a set with *start* and *end* pairs that define the session indexes of beginning and end within the matrix $E$.

Thirdly, it joins all sequences from events within the *StartEnd* set. Accordingly it derives the number of events for each session, the duration and obtains the final label and context for the session, by the following criteria.

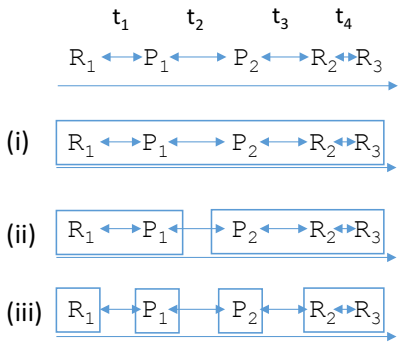



Figure 3.7 – Skill for (i) 200 seconds, (ii) 100 seconds and (iii) 60 seconds

In case the subset of events has more than one malicious label (line 10) it keeps the label with higher frequency or the first found (in case of equally distributed malicious labels). The same criterion is calculated by using the statistical *mode* to obtain a single context (line 15).

Accordingly, an example is given in Figure 3.7 for time duration between each events $t_1$, $t_2$, $t_3$ and $t_4$ and different *Skill* seconds. We can observe that for this example, an employee with low average time or high skills will lead to sessions with shorter sequences of actions, while a low-level skill will derive in longer sequences. This is an assumption under the basis that a low-skilled employee may take a longer time (e.g., type in, decide which operation to use and how) due to the lack of efficacy or experience in performing his role.

## 3.5 Anomaly behavior model

The threat scenarios used in this section concerned collected use cases from security groups [28] and previous malicious insider datasets adapted to simulate the interaction with cloud assets (e.g., [70], [113]). These examples are classified in content-based and event-based: where an employee goes rogue pursuing a sparse event concerning IP theft, sabotage or other. The second group of concern, studies the statistical information of the events, and analyses them following a graph dependency with respect to the assets in time.

The present work aims to tackle both of these groups of threats. Moreover, the anomaly behavior model describes the activities an employee performs with respect to the cloud assets. The profile for every employee under a specific role is performed by using the logged *username* or *hostname*. Therefore, anomalies performed by the attacker are considered by studying content-based and contextual features with respect to their historic data and possibly with the rest of the employees with the same `Role`. We cover the following threat anomaly cases, listed in Table 3.1.

Table 3.1 – Summary of anomalies concerning cyber behavior

| Anomaly | Acronym | Description | Example |
|---|---|---|---|
| Unusual list of tasks | SEQ | Intention to perform an activity to an asset that damages the integrity, confidentiality or availability of the asset | Data deletion |
| Unusual hour of access | WH | Intention to perform an activity to an asset outside the general working hours for the particular role | Activity at 3 a.m. |
| Unusual location of access | LOC | Intention to perform an activity to an asset from an unusual different source address | Connection from employee's house to cloud services |

**Unusual list of tasks** This behavior consists of an intent performed by an employee to pursue an unusual set of commands, such as elevation of privileges (i.e., obtaining further privileges with respect to the CSC's asset), storing remote information, creating new services (e.g., VM for further malicious activities), performing an unusual instruction to the Database Management System (DBMS). The characteristic for this anomaly consists in observing unusual commands.

**Unusual hour of access** This behavior consists of an activity performed by an employee different from normal working hours (e.g., between 7:00p.m. – 8:00a.m.). The working hours parameter is derived from the profile configuration, specifically the `Context` attribute. This relies on the fact that some privileged users, such as systems administrators, typically connect remotely to various systems outside office hours in the normal course of their daily activities.

**Unusual location of access** This behavior consists of an activity performed by an employee different from their everyday geographic location (i.e., usually from organization's location or other remote connection accepted by the organization). According to Moore et al. [90] in 15% of the cases they studied the insiders used remote access to attack.

## 3.6 Verification of the User Model

In the previous Sections, we have presented the ontology relationship for a `Profile` along with the anomalies to be studied. For simulating this model in a cloud scenario, an implementation in `Python` has been developed. The operation of this program is evaluated through random testing for each of the following modules, depicted in Figure 3.8.

The following paragraphs have per objective the validation of a set of invariants appertaining to the generated datasets from the user model. This is done by means of pre-/post conditions that are evaluated against the resultant datasets generated from the simulation.

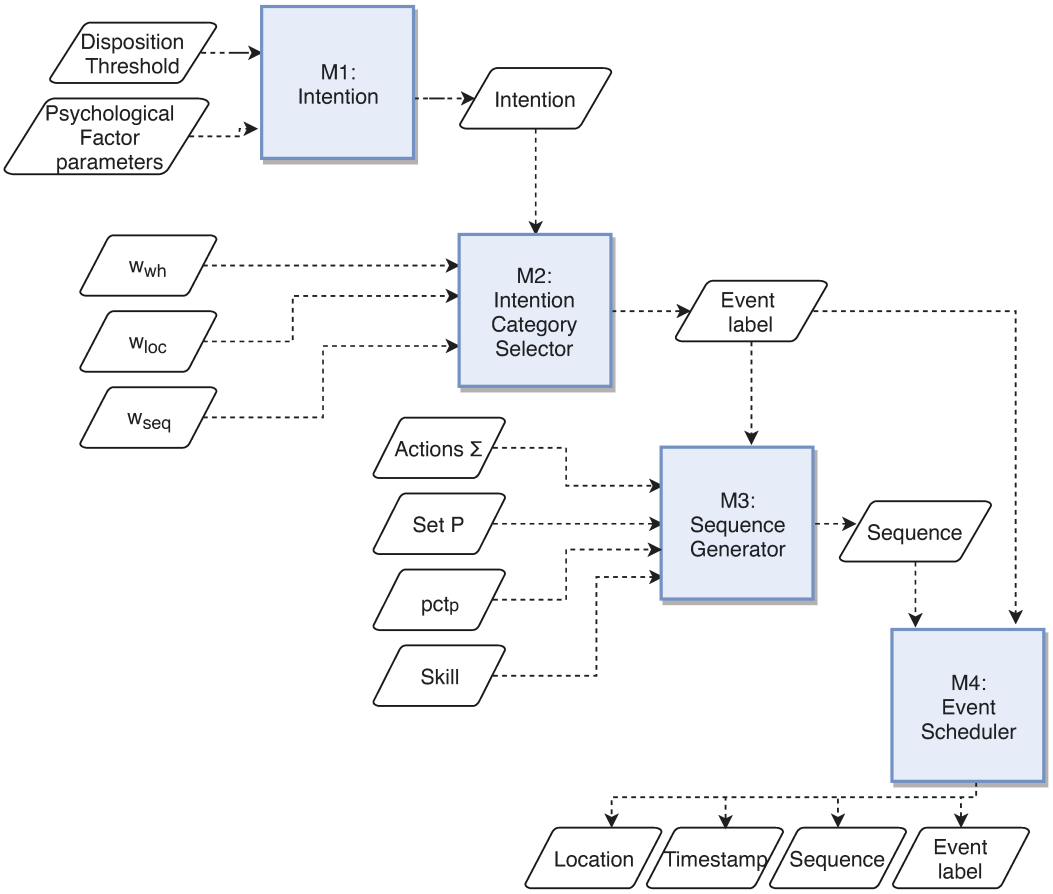



Figure 3.8 – Data flow of the simulation components

### 3.6.1 Verifying the intention selector module

This function corresponds to the **M1: Intention** in Figure 3.8 and it has relation with the employee's **Psychological factor** characteristics. The Figure 3.8 depicts inputs and outputs as

- Inputs
  - Disposition threshold constant $disposition\_thresh \in [0, 1]$
  - **Psychological factor** function $P_t : \mathbb{R} \to [0, 1]$
- Output
  - Intention $i_t \in \{0, 1\}$ where 1 is *perform a malicious activity* and 0 is *not perform a malicious activity.*

This function follows a time-dependent probabilistic approach where for each time step $t$ of the simulation, the intention of the employee is obtained when performing an `Event`. This is done through Equation 3.10.

$$i_t = \begin{cases} 1 & \text{if } p_t > disposition\_thresh \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

Consequently, let $n$ be the number of instances obtained, where the matrix $\mathbf{M1}_{n\times 3}$ of inputs and outputs consists in

$$\begin{bmatrix} disposition\_thresh_1 & p_1 & i_1 \\ \vdots & \vdots & \vdots \\ disposition\_thresh_n & p_n & i_n \end{bmatrix}$$

Let the average count of $x$ occurrences be defined as

$$C_x = \frac{1}{n} \sum_{j=1}^{n} [i = x] \tag{3.12}$$

whereas the average count of malicious occurrences is $C_1$. Therefore, the condition derived for this function is

**Pre-/postcondition 1** : if $disposition\_thresh \in [0, 1]$ then $C_1 \leq disposition\_thresh$

### 3.6.2 Verifying the intention category selector module

This function corresponds to the **M2: Intention Category Selector** in Figure 3.8 and it has relation with the election of the anomaly types, presented in Table 3.1. This is done according to the constants $w_{loc}$, $w_{wh}$ and $w_{seq}$, which are defined in the simulation's initial configuration. The Figure 3.8 depicts the inputs and outputs at step $t$ of the simulation as

- Inputs
  - Intention $i_t \in \{0, 1\}$
  - Weight for Unusual location of access anomaly (LOC) $w_{loc} \in [0, 1]$

  - Weight for Unusual hour of of access anomaly (WH) $w_{wh} \in [0, 1]$
  - Weight for Unusual list of tasks anomaly (SEQ) $w_{seq} \in [0, 1]$

- Output
  - Label of the event $l_t \in$ {LOC, WH, SEQ}

For $C_i$ with $i \in$ {LOC, WH, SEQ} correspond to the average occurrences for events WH, LOC and SEQ, respectively. Therefore, the conditions derived for this function are

**Pre-/postcondition 2** : if $w_{wh} + w_{loc} + w_{seq} = 1$ then $C_{wh} \leq w_{wh}$

**Pre-/postcondition 3** : if $w_{wh} + w_{loc} + w_{seq} = 1$ then $C_{loc} \leq w_{loc}$

**Pre-/postcondition 4** : if $w_{wh} + w_{loc} + w_{seq} = 1$ then $C_{seq} \leq w_{seq}$

### 3.6.3 Verifying the sequence generator module

This function corresponds to the **M3: Sequence Generator** in Figure 3.8 and it has relation with the employee's **Cyber factor** attributes. As mentioned in Section 3.3, these characteristics are derived by generating the sequences of actions the employee is going to execute. This is done by considering the following inputs, designated in the initial configuration for the simulation: (i) the set of actions the employee is allowed to execute, according to the cloud `Asset` and the `Role`'s policies; (ii) the predefined sequences of actions, related to the `Profile`; (iii) the percentage for the predefined sequences within the output matrix of sequences $\mathbf{S}_{n\times 1}$; (iv) the skill [1] (**Cyber factor**, detailed in Section 3.4.4) or level of sophistication to perform the actions to the particular `Assets`. The Figure 3.8 depicts inputs and outputs at step $t$ of the simulation as

- Inputs
  - Alphabet of actions, defined as $\Sigma$
  - Predefined sequences subset, defined as $P \subset \Sigma^*$
  - Predefined sequences presence percentage $pct_p \in [0, 100]$
  - Technical Skill $skill \in \mathbb{R}^+$
  - Event label $l_t \in$ {LOC, WH, SEQ, NORMAL}
- Output
  - Sequence of actions $s_t$ to be executed

Therefore, for the presence of predefined sequences in final output matrix [2], the conditions derived for this function are:

**Pre-/postcondition 5** : if $pct_p \in [0, 100]$ then $100 \cdot C_{\{p \in P\}} > pct_p$

[1] Defined as the average time in seconds an employee takes to execute a set of actions.
[2] Where $C_p$ is the average count of sequences $p$, as described in Equation 3.12.

### 3.6.4 Verifying the event scheduler module

This function corresponds to the **M4: Event Scheduler** in Figure 3.8 and it has relation with the employee's contextual behavior characteristics. These characteristics are derived by the modification of time and location variables, in case the *label* of the event (Equation 3.1) is WH or LOC. For this matter, the location position is mapped from the contextual information (i.e., `ip_source`) and is defined as a categorical variable $pos \in POS$, where $POS$ is a set of all possible countries' acronyms. In other words, it corresponds to the geographical origin from where the sequence of actions $s_t$ is executed (e.g., FR for France, RU for Russia). The Figure 3.8 depicts inputs and outputs at step $t$ of the simulation as

- Inputs
  - Sequence of actions executed, defined by $s_t$
  - Input location from where the event was executed $pos_{init\_t}$
  - Working hours context given by the role $ctx_t$
  - Event label $l_t \in$ {LOC, WH, SEQ, NORMAL}
- Output
  - Sequence of actions executed, defined by $s_t$
  - Output location from where the event was executed $pos_{out\_t}$
  - Timestamp of the event $ts_t$
  - Event label $l_t \in$ {LOC, WH, SEQ, NORMAL}

As mentioned, in case the `Event` label is *wh*, the resultant timestamp of the event has to be outside the hour range of the WH derived from the context $ctx_t$. Same modification methodology is treated with the `Event` label LOC, where the final location position of the `Event` $pos_{out\_t}$ has to be different from the initial $pos_{init\_t}$. Therefore, the conditions derived for this function are

**Pre-/postcondition 6** : if $l_t = wh$ then $ts_t \notin ctx_t$

**Pre-/postcondition 7** : if $l_t = loc$ then $pos_{init\_t} \neq pos_{out\_t}$

### 3.6.5 Validation criteria

Defining suitable criteria for dataset validation is a complex process, since there are no general methodologies in the literature [26]. While the insider threat simulation is implemented, some points of the proposed approach need to be validated before using them on real use cases. To this end, three validation criteria have been defined. The first two (namely, items 1 and 2) add an *a priori* degree of realism to detect plausible attacks, as the result of consulting with the CSC use cases in industrial reports. The third (namely, item 3) relies on a *posteriori* verification and proves the applicability of the proposed approach given the nature of the data for prediction or detection techniques.

1. **Similarity with respect to the average number of events per day**: industrial case studies indicate a number of actions an employee should do on a monthly basis. In this

case, for example an expert knows that a security administrator can initiate an action at any given time (e.g., 24x7 service availability) while a DBA `Role` should not initiate more than $N$ `Events` per month, that consider a database back-up `Sequence`. The generated `Events`, should be statistically based on *realistic* behaviors for every `Profile` entity and each threat scenario. Such statistical data can be either provided by an "oracle" aware of the activities for each `Role`, or from traces of real case studies for later extrapolation.

2. **Sequences' realism**: the pseudo-random generated set of `Sequences` for each `Event`, should be validated to make sense in the context of the `Permission-Asset` tuple. In other words, independent actions to a given `Asset`, for example, "data elimination" or "tampering" from a database, might not have a pre-defined order. In the case of other tuples, such as actions to a VM `Asset`, it might be intuitive to generate `Sequences` with a given order (i.e., an action of shutting a VM down cannot be followed by any other operation that assumes the VM is operational). The latter means that the set of invariants from the second group includes the ordering of the actions performed, i.e., action "B" in a `Sequence` cannot be executed by a given `Profile` unless the action "A" has been processed.

3. **Anomaly detection techniques benchmarking**: For an accurate prevention of this threat, proper anomaly detection benchmarking can be performed. For this matter, the dataset should contain "well distributed" labeled `Events` or its technique should recognize possible label imbalance (under the assumption that malicious `Events` are less frequent than *normal*). This is relevant at the moment of experimenting with detection techniques such as supervised machine learning models, as they can try to fit *anomalies* with *normal* events.

## 3.7 Experimental results of the dataset generation

The simulation methodology was implemented using an ABM, where an agent has a psychological and cyber behavioral factors for every iteration step in the simulation. The benefit of an ABM strategy is that it provides a mean to represent complex adaptive behaviors by focusing on the attributes of the individual heterogeneous employees' entities (agents) and how they interact within a larger system. Additionally, it allows the possibility to simulate entities simultaneously interacting with the different CSC's `Assets` at the same time.

The implementation assumes that an organization is made of a certain number of heterogeneous employees. As mentioned in Section 3.4.1 and detailed in Algorithm 2, those employees have the potential to become a malicious insider based on a combination of emotional, rational and social factors affecting their disposition. We designed different scenarios where we derived three profiles varying: Equations 3.2 (Rescorla-Wagner), different weights for the Expectation Equation 3.3 and different reward_value, risk_tolerance parameters within the Equation 3.10. The presentation of these results is gathered in the following Section.

### 3.7.1 Experimental results

As we represent each profile's behavior following a role-based approach, we utilize as an example the role of a DBA, defined as a user in charge of administrative actions towards the database, such as installation, patching and upgrade of the database. This includes the ownership of all objects of the database and the ability to create and modify roles, users and data files.

The aim of this set of experiments is to the benefit of observing different scenarios for posterior detection analysis. For that matter, the agent's characterization is then derived from the variables from two different groups. The first relies on the input variables for the disposition function (Table 3.2), and the second group (Table 3.3) is derived with respect to cloud-related, contextual and cyber parameters (e.g., number of agents in simulation, roles for each agent, set of actions).

The actions for these experiments were taken from the set $A = \{C, R, U, D\}$. Note that in the implementation in Chapter 5, these actions can be fictitious or real commands towards an implemented asset.

Table 3.2 – Sweep parameters for Disposition function in simulation

| Parameters | DBA 1 | DBA 2 |
|---|---|---|
| Rational probability $V_{reward}$ | 0.5 | 0.5 |
| Rational probability $V_{risk}$ | 0.1 | 0.5 |
| Affective probability $\omega_{init}$ | 0.001 | 0.001 |
| Affective probability $\omega_{actual}$ | 0.3 | 0.15 |
| Affective probability $\omega_{historic}$ | 0.9 | 0.7 |
| Disgruntled threshold ($\tau$) | 0.2 | 0.5 |

As mentioned in Section 3.4.1, each `Profile`'s `Role` has pseudo-random, predefined and hybrid group of generated `Sequences` followed by a *normal* behavior. The following illustrations represent the used examples of the events a DBA can perform and, therefore, define the DBA *normal* behavior:

- A DBA in a working day, logs into the DBMS and enrolls a new user with write permission over a database.
- A DBA regularly works remotely on Wednesdays, logging into the DBMS from location $a$, while the rest of the week from location $b$.

On the other hand, the following expressions can be represented as generated `Sequences` for an *anomalous* DBA behavior:

- LOC anomaly: A DBA logs in from a public IP that does not belong to the company and performs a *Sequence* of actions. In this case, the insider may connect to any machine via port 3389 (RDP), 23 (Telnet) or 22 (SSH).
- WH Anomaly: A DBA, logs in and performs numerous `Sequences` of actions on the database.
- SEQ Anomaly: A DBA, logs in and performs different `Sequences` of actions on the database than its normal behavior.

Three types of `Profiles` (DBA 1, DBA 2) with the same DBA `Role` have been outlined, differentiating them by a created **Cyber factor** named "skill level" as described in Table 3.3. This factor defines the time taken to perform a `Sequence` of actions with low, medium and high skills and prompts the `Sequences`' length. We also have modeled the three profiles with the **Psychological factor** derived from different parameters to be malicious in Table 3.2.

Such cases are depicted in Figure 3.5, where the profile generation is performed under the assumptions given by the industrial case studies [74]. Additionally, in our example the generation of `Events` is treated by the "No. Monthly events" which we have settled at 10 `Events` per day.

Table 3.3 – DBAs Parameters for cloud-related, contextual and cyber factors

| Parameters | DBA 1 | DBA 2 |
|---|---|---|
| IP 192.168.1.* | .100 | .110 |
| Location | France | France |
| WH | 9am-6pm | 9am-6pm |
| Skill level | 30 (high) | 60 (med) |
| No. Daily Events | 10 | 10 |

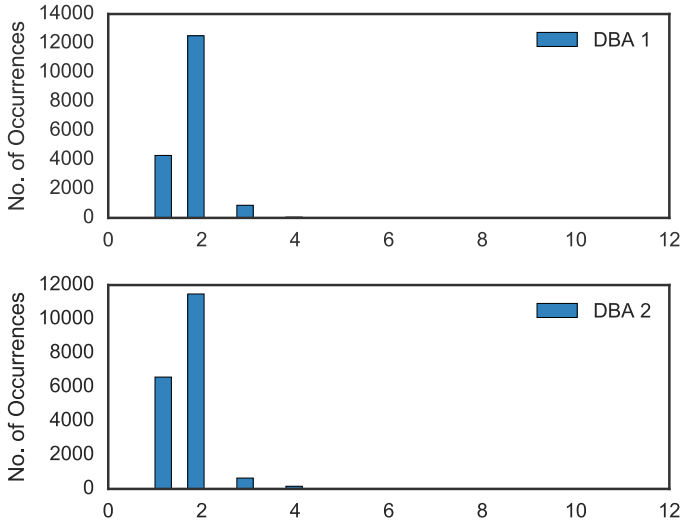



Figure 3.9 – Histogram example of sequences' length for Profile with high, medium and low skills

Along with the implementation previously described in this section, a scalability benchmark was performed and exposed in Tables 3.4 and 3.5. Table 3.4 depicts different the simulation for different time duration (in months) with a fixed number of employees (agents). This variable is compared with the time the simulation takes to finish (calculation time) and the amount of information in MBytes for the generated output.

Table 3.4 – Benchmark for simulation time in months

| Simulation time (months) | Calculation time (minutes) | Storage capacity (MBytes) |
|---|---|---|
| 8 | 7 | 26 |
| 12 | 11 | 40 |
| 15 | 13 | 50 |
| 20 | 19 | 66 |

Table 3.5 – Benchmark for number of users

| Simulated users | Calculation time (minutes) | Storage capacity (MBytes) |
|---|---|---|
| 6 | 2 | 300 |
| 12 | 8 | 1000 |
| 24 | 27 | 4300 |
| 36 | 59 | 8700 |

## 3.8 Discussion

Cloud computing security is ripe with new opportunities for future research, including cloud-related insider threats. As mentioned previously, we do not believe the nature of the insider will change due to CC's impact, but the opportunities for attacks will broaden.

The validation of human behavior and social interactions poses a challenge because of the random nature of the system being simulated and the difficulty of gathering empirical evidence due to this variability [123]. From a theoretical validity point of view, the agent-based model-based approach is grounded in renowned relevant theories. From an empirical point of view, we utilized statistical and structural information from industrial case studies, gathered primarily by the CERT [74].

Also, foundational work in risk management suggests that if an individual has motive, capability, and opportunity, then they are likely to conduct an attack. However, a crucial question here is, what constitutes as "enough" motive, or "enough" capability? Likewise, somebody may well exhibit all these, and yet still choose not to attack. Much previous literature also discusses the concept that if an individual is disgruntled then they may choose to act out.

## 3.9 Conclusion

In this chapter, we have addressed some important research topics regarding insider threat resulting in an enriched user model for further simulation and dataset generation. Accordingly, we outline the design of the aimed synthetic data, while discussing cloud-based indicators, and psychological-technical human factors, finally proposing a socio-technical approach to insider threats.

Additionally, in Sections 3.4 and 3.6, the model for deriving insider threat scenarios was introduced, along with the simulator's pre-/post conditions description.

This checking process was designed to test whether or not the agent-based model was capable of reproducing the conditions of the preliminary conceptual model. The methods for checking the simulation's pre-/post conditions included iterative programmatic testing ("debugging") and exploratory analysis of the simulated data obtained from execution of the model. The simulation analysis included a parameter sweep, in which the model input variables were adjusted systematically followed by an individual time series experiment.

Additionally, we studied the performance of the ABM implementation, detailed in Table 3.4 and Table 3.5. The first related to the calculation time taken to provide the dataset scenario, as we varied the simulation duration in a range of [8,20] months while setting the number of users to three. The second correlates the same variables when the number of users is increased and the simulation time is fixed at 5 months.

The proposed model provides a promising exploration characterization of the insider threat,

from an ABM model paradigm. The results exposed allow the use of this software to derive different characterizations, not only considering distinct psychological attributes but also benefiting itself from different cyber behaviors.

As a conclusion, we can say that the results obtained will be very useful for intrusion detection techniques and, in particular, for these working on malicious insider threats.

Results with regards of simulation and calculation time, aside with the dataset sizes, reveals the outcome behaves scalable enough, where simulated data equivalent to a year takes only 11 minutes of simulation time.

Chapter 4

# Anomaly-based Detection Framework

**Contents**

## 4.1 Introduction

In this Chapter, an overview of the proposed framework is first given, and each component is described in greater detail. Moreover, the primary objective is to describe the most relevant aspects to consider at the moment of distinguishing anomalies with a clustering technique for established insider threat scenarios. These scenarios are studied as a result of the implementation of an insider threat model, described in Chapter 3. This model is composed of a combination of probabilities to be malicious, and different actions towards cloud assets. This detection scenario is complicated, where defining possible precursors for an insider threat in terms of observable traces of the employees' actions, and integrating these contributions in an analytic model is a significant challenge.

There are numerous alternative ways of doing this, and therefore it cannot be in the scope of this thesis to evaluate them all. By considering this matter, we have defined some specific objectives based on the intent to solve use-case scenarios.

As discussed throughout this work, we denote the realistic assumptions under which our solution is made.

- First, by definition, an insider has legitimate access to the organization's resources. As such, it is difficult to define the necessary criteria to discern legitimate activity from that which is non-legitimate. Regardless of how effective anomaly detection is, "anomalous behavior can never equate exclusively to misuse or lack of legitimacy".

- Second, regarding these insider's activities, as many others have stated [76], [79], [106], [126], we consider that normal data constitute an overwhelmingly large portion[1] of the collected data in a reasonable mid-term time window, while anomalies either belong to small or sparse clusters.

- Third, compared to the traditional approaches of post-attack analysis and subsequent change of policy, predicting threats from data offers the benefit of continuous and an online evaluation [12], [49]. Therefore, CSCs need reacting in a timely manner to these threats, as insiders could comprise the availability, confidentiality or integrity of their assets.

Assumptions listed above allow obtaining the following main contributions of this Chapter, namely: 1. A detection technique and corresponding experimental results; 2. A monitoring solution for IaaS-based services regarding the detection of malicious insiders; 3. A list of attributes that should be considered for malicious activities in cloud applications under an IaaS model.

The structure of the chapter begins with the preliminaries (Section 4.2 and the main concepts related to anomalies, the data representation and the algorithms that served as inspiration for the anomaly detection framework design. Later, the main solution is presented (Section 4.3), concerning the data analysis of the different attributes, and the main contribution of the present work. Discussion (Section 4.4) gives detailed insights on the results found while the last part of the chapter (Section 4.5) ends with the conclusions.

[1] Usually considered bigger than 80% for the presented experimental work, but without the loss of generality it can be adjusted to other values.

## 4.2 Preliminaries

### 4.2.1 Anomalies

Many names are used as synonyms for anomalies in the literature, including outliers, abnormalities, discordants or discords [2]. Chandola et al. [23] defined an anomaly as

*Patterns in data that do not conform to a well-defined notion of normal behavior.*

Chandola et al. make the distinction between noise and anomaly, with the latter being outliers of interest to the subject analyzing the data whereas the former is not. The only effect noise has, is to hinder the data analysis task.

Given that the present work focuses on the automated detection of anomalies, rather than on their interpretation by analysts, such distinction is not considered relevant; therefore, "anomaly", "noise" and "outlier" are understood here as synonyms, unless explicitly stated otherwise.

Anomalies can be categorized as follows [23].

- Point anomalies: data instances that are considered anomalous with respect to the rest of the data. They are the simplest form of anomaly and the focus of most research on anomaly detection.

- Contextual anomalies: data instances that are only considered anomalous in a specific context. This context must be induced from the data, and thus it requires each instance to be defined using both:

  - Contextual attributes, which are used to determine the context of that instance; e.g., temporal, geographical characteristics, for a character sequence.
  - Behavioral attributes, which determine the non-contextual properties of the instance, e.g., the employee's activity towards a cloud asset.

### 4.2.2 Data collection and representation

#### 4.2.2.1 Data collection

As mentioned in Chapter 2, by means of detecting anomalies in a targeted environment, a detection system may collect data from different points of observations (e.g., hypervisor-based, network-based or host-based).

The proposed data collection is enriched by comparing two different points of data capture: a profile-based view from the local network of the company, and a cloud-end view that analyses data from the services with whom the clients interact. The latter is considered from an end-user or CSC perspective, i.e., we assume we do not have a sensor in a CSP proprietary observation point, but instead we do where the CSC has access.

When the activity of a particular type described above is unusual in a way or to a degree known to correlate with malicious insider actions, we can treat these features as relevant indicators.

#### 4.2.2.2 Representation of user sequences of actions

There are two major challenges in sequence classification. First, most of the classifiers, such as decision trees and ANNs, can only take input data as vectors of features. Second, even with various feature selection methods, we can transform a sequence into a set of features, the feature selection is not a trivial task. The reason is because the dimensionality of the feature space for the sequence data can be very high, and the computation can be costly [138].

At the moment of working only with system call sequences, the various implementations differ in how data are represented. In general, these representations can be grouped into two categories based on their feature extraction methods: (i) Frequency-based methods (ii) Sequence-based methods.

**Frequency-based** methods rely on the number of occurrences of each system call. For example, using a "bag of words" representation (which is commonly used in text classification), a system call anomaly detection can be mapped into this representation.

Thus, previous works [1], [5], [35], [49], [67], [103], [116], [127], [137], [141] related to frequency-based approach mostly utilize the "bag of words" representation or considering the number of occurrences in a arbitrary time window, since it is a manner of transforming the sequences into a propitious vector input for machine learning algorithms.

For example, Gavai et al. [49] utilized features related to email (e.g., number of emails sent in a day) and web usage (e.g., average time spent on websites), log-in and log-out (e.g., number of log-ins) to detect anomalous activities. Instead of only counting the number of occurrences, some approaches improve detection by applying a ranking, based on the relative order of frequency values [130]. Also, following the frequency-based approach, Parveen et al. [103] utilized a compression-based frequent pattern discovery in order to propose a graph-based anomaly detection. Song et al. [124] used the system level data with a Windows sensor, to further detect user behavior bio-metrics with a Gaussian Mixture Model (GMM). For this, they transformed their data into vectors of the number of occurrences (e.g., number of unique processes, number of user touches, number of files touched). Senator et al. [116] also followed a graph-based anomaly detection scheme, by treating proxy, email and LDAP logs as the number of occurrences, and the percentage of the logs dedicated to emails, attachments on sent emails, among others.

**Sequence-based** methods use the order of the sequences as information. They can be divided into three large categories [138].

1. The first category is *feature-based* classification, which transforms a sequence into a feature vector and then applying conventional classification methods. Feature selection plays an important role in this kind of methods.

2. The second category is *sequence distance-based* classification. The distance function which measures the similarity between sequences determines the quality of the classification significantly.

3. The third category is *model-based* classification, such as using HMM and other statistical models to classify the sequences (e.g., [108]).

#### 4.2.2.3 Network-based representation

Network core equipment such as routers, switches, firewalls, all have the ability to collect the network traffic passing through which, traditionally, are considered essential data sources for detecting intrusions. Moreover, some works have identified such network logs' great potential for addressing insider threats (e.g., [82]).

The proposed work has focused on retrieving the most relevant attributes from a Deep Packet Inspection (DPI) tool, in order to reconstruct the network behaviors and interaction patterns, through the parsing of the packet headers, and obtaining relevant statistics through the algorithm's analysis.

The proposed set of attributes is the result of a monitoring implementation (Chapter 5).

#### 4.2.2.4 IaaS-based representation

As mentioned, the proposed work has focused on IaaS assets, such as virtual machines and images and network properties within the cloud implementation, correspondent to a CSC. In order to correctly detect anomalies towards the cloud, and in consideration of the usually restrictive access CSPs grant to CSCs, it is desirable to characterize the behavior of the usage of these assets, through monitoring and collection of propitious information related to them.

Hence, the proposed set of attributes, whose implementation and calculation is specified in Chapter 5, along with their analysis in this Chapter, advises a sufficient group of features, capable of adequately representing the characterization of the normal usage of them, along with showing when anomalous activities take place.

### 4.2.3 Clustering methods

We remind from Chapter 2 that the purpose of the clustering techniques is to divide a dataset into distinct groups, or clusters. Opposite to categorization, which aims to sort the data points into predefined groups, clustering aims to find unknown structures in the dataset and do knowledge discovery. Since clustering deals with unlabeled data, these techniques are unsupervised.

This approach properly treats the proposed challenges and is chosen in the present work by means of detecting anomalies for the insider threat.

**Growing Neural Gas (GNG)** From its first presentation by Fritzke [43], this algorithm has been extended for different purposes in the past years. Generic contributions (e.g., [50]) have been done in the data mining field by means of experimentation with different machine learning datasets (e.g., UCI datasets [15]), and also it has been proposed as a solution in specific applications domains. This algorithm has proven to be an efficient solution in several application fields such as image recognition (e.g., MRI [6], Robotic 3D motion images [132]) and anomaly detection (e.g., [13], [126]).

In more detail, GNG is an incremental self-organizing approach which belongs to the family of topological maps such as SOM [73] or NG [85]. It is an unsupervised clustering algorithm capable of representing a high dimensional input space in a low dimensional feature map and discover topological relationships of the data.

This technique presents many advantages to other unsupervised clustering techniques. With respect to its predecessors, SOM and NG, it does not need to fix the graph size in advance. Additionally, it allows the continuous learning and growth of the network automatically. This

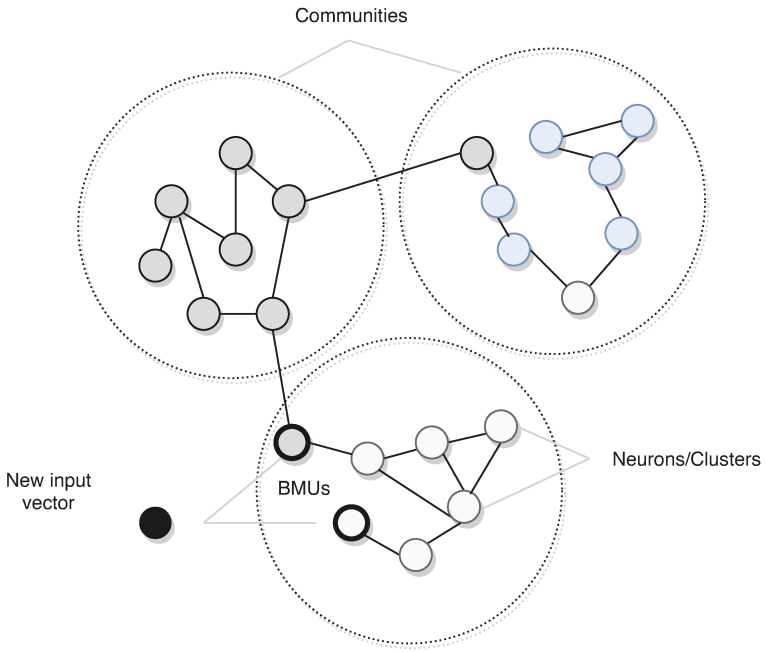



Figure 4.1 – 2D Graph representation of a GNG network

means that it can add neurons into its map space whenever the input approximates the input space more accurately.

As seen in Figure 4.1, it is meant to be a 2D representation of a multi-dimensional dataset. In this 2D representation, each of the data vectors' inputs, e.g. collected data from the cloud, is mapped to one of the neurons on the 2D grid. Most importantly, similar (high-dimensional) inputs are mapped to the same neuron, or at least the same region in space.

In time the algorithm constructs a graph of neurons, in which each neuron $n$ has its associated weight or prototype $w_n$. Prototypes can be seen as positions of the neurons correspondent to the input space of the absorbed input vectors. Pairs of neurons are connected by edges (links), which are not weighted. The purpose of these links is to define a topological structure. These links are temporal in the sense that they are subject to aging during the iteration steps of the algorithm and are removed when they become "too old".

The original GNG can be resumed in 4 dominant procedures: (i) **Learning**, (ii) **Insertion**, (iii) **Deletion** and (iv) **Stopping**. The overall process is detailed in Algorithm 5 and the notations used throughout the Chapter are described in Table 4.1.

The procedure of Algorithm 5 starts with two neurons, and as a new data point is available, the nearest neuron ($bmu_1$) and the second-nearest neuron $bmu_2$ are identified and linked by an edge; later on the nearest neuron, and its topological neighbors are moved towards the data point (**Learning**).

In the original algorithm, the process of finding the two nearest neurons $bmu_1$, $bmu_2$ (winner and second winner) is done by comparing the input vector $x$, $x \in \mathbb{R}^d$, where Euclidean distance is used as the metric. An extension to this metric is presented in Chapter 6 where additional distance metrics have been applied by means of analyzing their effect in the algorithm.

Each neuron has an accumulated error variable. Periodically, a neuron is inserted (Algorithm 6) into the graph between the neurons with the largest error values (**Insertion**).

Table 4.1 – Notations for the proposed and compared algorithms

| Notation | Range | Description |
|---|---|---|
| $\mathcal{DS}$ | | Stream of $n$-dimensional data vectors |
| $x(i)$ | $\in \mathbb{R}^+$ | Value for feature $i$ in vector $x$ |
| $N$ | $\in \mathbb{N}$ | Set of neurons |
| $N_u$ | $\in \mathbb{N}$ | Set of neurons connected to neuron $u$ |
| $w_u$ | $\in \mathbb{R}^d$ | Prototype $w_u = (w_u^1, w_u^2, ..., w_u^d)$ of neuron $u$ |
| $w(i)$ | $\in \mathbb{R}^+$ | List of prototype for feature $i$ for all neurons in $\mathcal{G}$ |
| error$(u)$ | $\in \mathbb{R}^+$ | Local accumulated error variable of neuron u |
| $\mathcal{E}$ | $\in \mathbb{R}^+$ | Set of $l$ errors for neurons $\{u_1, u_2, \ldots, u_l\}$ |
| $bmu_1$ | $\in \mathbb{N}$ | BMU (nearest neuron to the $x_i$ input) |
| $bmu_2$ | $\in \mathbb{N}$ | Second BMU |
| $H$ | $\in \mathbb{N}_0$ | Set of hits for all neurons $N$ |
| $H_u$ | $\in \mathbb{N}_0$ | Number of hits for neuron $u$ |
| $\alpha_1$ | $\in [0, 1]$ | $bmu_1$ (the nearest neuron) adaptation factor |
| $\alpha_2$ | $\in [0, 1]$ | $bmu_1$'s neighbor adaptation factor |
| $\beta$ | $\in [0, 1]$ | Global error factor update |
| $\lambda$ | $\in \mathbb{N}$ | Cycle interval between neuron insertion |
| $\eta$ | $\in \mathbb{N}$ | Cycle interval between neuron deletion |
| $age_{max}$ | $\in \mathbb{N}$ | Oldest age allowed for an edge |
| $T_{server}$ | $\in \mathbb{R}^+$ | Threshold for server severity in $\mathcal{G}$ |
| $T_{image}$ | $\in \mathbb{R}^+$ | Threshold for image severity in $\mathcal{G}$ |
| $T_{network}$ | $\in \mathbb{R}^+$ | Threshold for network severity in $\mathcal{G}$ |

Neurons can also be removed if they are identified as being superfluous (**Deletion**). Finally, the original algorithm finishes learning (**Stopping**) when the stopping criterion is achieved.

The presented processes are described in detail, denoting how our approach has adapted and extended them for anomaly detection purposes for the insider threat.

**Learning** The original GNG of Algorithm 5 learns through fixed parameters $\alpha_1$ and $\alpha_2$ for $bmu_1$ and $bmu_2$, respectively, as shown in Equation 4.1. This is a disadvantage that makes the network less adaptive to the incoming data. In other words, every time a neuron "wins", it will update its prototype ($w_i$ in Equation 4.1) by a fraction of the absorbed input.

By means of utilizing similar approaches as in [44], in the proposed algorithm the learning rates are dependent of the number of times the neuron has won. This corresponds to the following definitions

$$\begin{aligned} w_u &= w_u + \alpha_1 * (x - w_u) \\ w_i &= w_i + \alpha_2 * (x - w_i), \forall i \in N_u \end{aligned} \tag{4.1}$$

As said, the learning rate determines the extent to which the winner and the neighbors of the winner are adapted towards the input signal.

In this work, we adopt an adaptive learning rate over time presented by [44] as

$$\begin{aligned} \alpha_1 &= \frac{1}{H_{bmu_1}(t)} \\ \alpha_2 &= \frac{1}{100\, H_{bmu_1}(t)} \end{aligned} \tag{4.2}$$

In this case, time parameter $H_{bmu_1}(t)$ represents the number of input signals for which this particular neuron has been a winner thus far, at time step $t$. The scheme considered follows the K-means approach, in the sense that every neuron is always the exact arithmetic mean of the input signals it has been a winner for [44]. This scheme is adopted because it makes the position $w_n$ of the neuron $n$ more stable by decreasing the learning rate when it becomes a winner for increasing input patterns.

To consider a broader set of learning rate approaches for the best suit of the insider threat scenario, an additional adaptive approach has been included in the analysis. This adaptive learning model proposed in [126] considers the Equation 4.2 presented by [44] and extended it as

$$\begin{aligned} \alpha_1 &= \frac{1}{H_{bmu_1}(t)} \cdot e^{\frac{\|\mathbf{x_i}-\mathbf{w_1}\|}{T_1}} \\ \alpha_2 &= \frac{1}{100 \cdot H_{bmu_1}(t)} \cdot e^{\frac{\|\mathbf{x_i}-\mathbf{w_n}\|}{T_1}}, \forall n \in N_u \end{aligned} \tag{4.3}$$

where $T_i$ is a similarity threshold, also presented in [44]. For classification purposes, the proposed model in [126] adds an exponential decay function to the existent in Equation 4.2.

**Algorithm 5** Original GNG [43]

**Input:** $\mathcal{DS}$ and model params $\mathcal{M} = (\lambda, \beta, \alpha_1, \alpha_2)$
**Output:** Neurons $\mathcal{N} = \{u_1, \ldots, u_n\}$ and prototypes $\mathcal{W} = \{w_1, \ldots, w_n\}$

1: Initialize two neurons with prototype vectors $w_1$, $w_2 \in \mathbb{R}^d$
2: **for all** x $\in \mathcal{DS}$ **do**
3: Find 1st neuron (winner):
$bmu_1 \leftarrow \text{argmin}_{bmu_1 \in N} \|x - w_{bmu_1}\|$
4: Find 2nd nearest neuron (second winner):
$bmu_2 \leftarrow \text{argmin}_{bmu_2 \in N \setminus bmu_1} \|x - w_{bmu_2}\|$
5: Modify the age of all edges emanating from $bmu_1$
6: Add the distance between $x$ and $bmu_1$ to error variable:
$error(bmu) \leftarrow error(bmu) + \|x - w_{bmu_1}\|_2$
7: Move $bmu_1$ and its direct neighbors:
$w_{bmu_1} \leftarrow w_{bmu_1} + \alpha_1 \cdot \|x - w_{bmu_1}\|_2$
$w_i \leftarrow w_i + \alpha_2 \cdot (x - w_i)$, $\forall i \in N_u$
8: **if** $i$-th x data is an integer multiple of $\lambda$ **then**
9: InsertNeuron (Algorithm 6)
10: **end if**
11: Delete each isolated neuron
12: Decrease the error of all neurons: $\mathcal{E} \leftarrow \mathcal{E} - \beta \cdot \mathcal{E}$
13: **end for**

**Insertion** As seen in Algorithm 5, a new neuron is inserted every $\lambda$ steps. Authors of [126] added a similarity threshold parameter $T_i$ that compares the BMU's distance with all its neighbors, and determines whether a neuron should be inserted following the original procedure of Algorithm 6.

**Algorithm 6** Original GNG [43] Neuron Insertion

**Input:** Graph $\mathcal{G}$
**Output:** Updated Graph $\mathcal{G}$

1: Find neuron $q$ with the maximum accumulated error, $error(q)$
2: Find the neighbor $f$ of $q$ with the largest accumulated error
3: Add a new neuron $r$, half-way between neurons $f$ and $q$:
$$w_r = 0.5 \cdot (w_f + w_f)$$
4: Insert edges connecting the new neuron $r$ with neurons $q$ and $f$
5: Remove the original edge between neurons $q$ and $f$

The similarity threshold can be calculated in two ways. If the neuron has direct topological neighbors, the threshold is updated like so in Equation 4.4, by using the maximum distance between $bmu_1$ and its neighbors.

$$T_i = \underset{n \in N_i}{\operatorname{argmax}} \|w_i - w_n\| \tag{4.4}$$

If the neuron $n$ has no neighbors, $T_n$ is updated as the minimum distance of neuron $n$ and all other neurons in the graph $\mathcal{G}$, like so in Equation 4.5:

$$T_n = \underset{n \in N_i \setminus \{i\}}{\operatorname{argmin}} \|w_i - w_n\| \tag{4.5}$$

### 4.2.4 Community detection

When we have a set of feature vectors learned by a clustering algorithm such as the mentioned GNG, we obtain a graph $\mathcal{G}$, as seen in Figure 4.2, in which each neuron represents a cluster of inputs, and edges represent the distance (e.g., Euclidean) between these clusters. Now we would like to know the general structure of this graph, i.e., $\hat{\mathcal{G}} = \operatorname{argmax}_p (\mathcal{G}|\mathcal{D})$ where $\mathcal{D}$ is our input vector distribution. Community detection has been studied as the graph partitioning in computer science for decades and remains quite challenging. Algorithms to detect reasonably good quality communities have been proposed and improved extensively, especially in recent years, such as Girvan Newman algorithm, spectral clustering, random walk, modularity optimization and statistical inference [56].

This graphical model is used to interpret the clustering structure, based on the topology to make anomaly predictions.

### 4.2.5 Evaluation metrics

Intuitively, the goal of clustering is to assign input vectors that are similar to the same cluster, and to ensure that vectors that are dissimilar are in different clusters. Hence, by performing an unsupervised organization of the data, there exists an underlying *ground-true* set of clusters, and the set of clusters found by our algorithm.

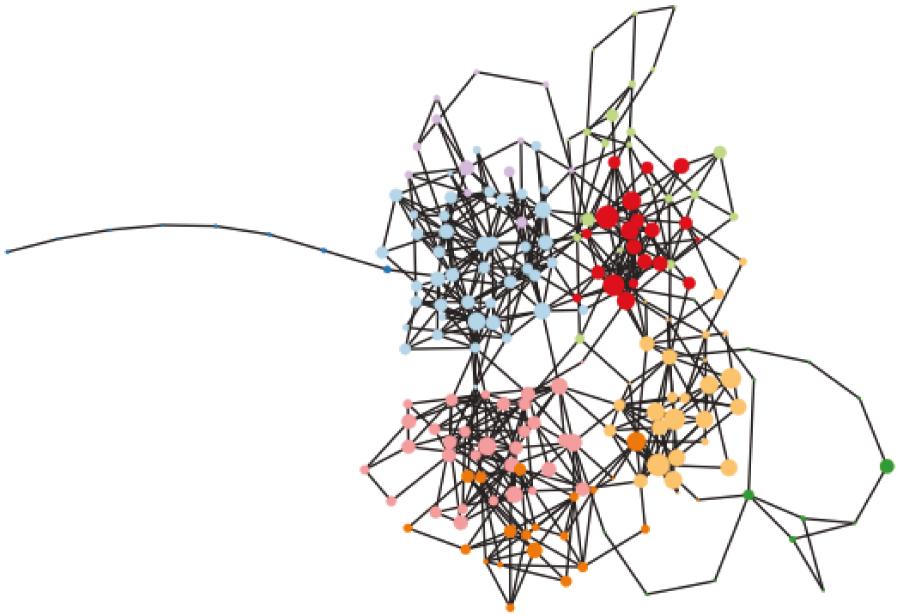


Figure 4.2 – GNG resultant topology for communities in simulated data where each neuron represents a cluster (size proportional to the number of data inputs within it) and each color represents a community

Therefore, in order to assess the quality of the clustering algorithm there has been chosen two of the most utilized metrics in the literature (e.g., [50]), namely internal and external, which evaluate the algorithm with different scopes.

The internal set of metrics intend to give information on the quality of the clustering process, with respect to the similarity between clusters. The external criteria on the other hand, intend to assess the quality of the algorithm through the comparison of the final clusters' topologies, with respect to the *ground-truth*. The latter, though completely left out in the anomaly detection process, is used by means of stating the clear usability, advantage and validation of our proposed algorithm. Each of the metrics proposed for evaluation, are detailed in the following sections.

#### 4.2.5.1 Internal criteria

Although internal criteria give no information with respect to the ability to predict and correctly separate data with respect to its correspondent labels, the internal analysis of the final clusters is a matter of validation to compare clustering algorithms, by means of studying how similar are intra and inter cluster data points, by the two following metrics.

**Silhouette coefficient** This metric [111] is widely used (e.g., [126]) in the context of clustering techniques and is based on the co-variation over the intra-separation and inter-separation distance from a clustering process. In other words, it characterizes the heterogeneity and isolation properties of the clusters.

$$S(C_i) = \frac{\sum_{v \in C_i} S_v}{|C_i|} \tag{4.6}$$

Where,

$$S_v = \frac{b_v - a_v}{max(a_v, b_v)} \tag{4.7}$$

In this case, $a_v$ is the average distance between vertex $v$ and all the other vertices in the same cluster as it is, and $b_v$ is the average distance between $v$ and all the vertices in the nearest cluster that are not $v$'s. The silhouette index for a given cluster is the average value of silhouette for all its member vertices. The silhouette index can assume values between -1 and 1, with a negative value being undesirable, as it means that the average internal distance of the cluster is greater than the external one.

**Modularity** This metric has been proposed through many definitions in the literature (e.g., [94]). One of the most utilized is the Louvain method [11] which is a validation metric for topological clustering. Modularity states that a good cluster should have a bigger than expected number of internal edges and a smaller than expected number of inter-cluster edges when compared to a random graph with similar characteristics.

In particular, the Louvain method of community detection is an algorithm for detecting communities in networks that relies upon a heuristic for maximizing the modularity. The method consists of repeated application of two steps. The first step is a "greedy" assignment of neurons to communities, favoring local optimization of the modularity. The second step is the definition of a new coarse-grained network in terms of the communities found in the first step. These two steps are repeated until no further modularity-increasing reassignments of communities are possible.

The Louvain method achieves modularities comparable to pre-existing algorithms, typically in less time, so it enables the study of much larger networks. It also generally reveals a hierarchy of communities at different scales, and this hierarchical perspective can be useful for understanding the global functioning of a graph.

#### 4.2.5.2 External criteria

The proposed algorithm for the insider threat detection is to be evaluated using three performance measures.

**Confusion Matrix** The confusion matrix [125] is commonly used to determine how well a classification model performs. It provides the information about the actual normal and abnormal number of instances, and the number of normal and abnormal instances in the analyzed results. Table 4.2 shows the confusion matrix for a two-class classifier. "True (T)" indicates that the prediction is correct, and "False (F)" is incorrect. "Positive (P)" is used to indicate an *abnormal* class and "Negative (N)" a *normal* class.

Consequently, there are four kinds of data in the confusion matrix to show the correct and incorrect predictions of the two classes: 1) True Positive (TP), 2) False Positive (FP), 3) True Negative (TN) and 4) False Negative (FN). For example, TP indicates the number of *abnormal* instances that are predicted correctly.

**Rate Measures** The basic evaluation measures are derived from the information that the confusion matrix provides. The definitions and the formulations of these evaluation measures are explained below.

Table 4.2 – Definition of confusion matrix

| ACTUAL | PREDICTED | |
|---|---|---|
| | Positive | Negative |
| Positive | TP | FN |
| Negative | FP | TN |

The *Precision* is the percentage of the correctly predicted anomalies (TP), computed over all predicted anomalies.

$$Precision = \frac{TP}{TP + FP} \tag{4.8}$$

The *Recall* is the percentage of the correctly predicted anomalies (TP) over all the actual anomalies. Recall is also known as the true positive rate or sensitivity.

$$Recall = \frac{TP}{TP + FN} \tag{4.9}$$

The *False Positive Rate (FPR)* is the percentage of the normal cases that are incorrectly predicted as anomalies (FP) over all the actual normal cases.

$$FPR = \frac{FP}{TN + FP} \tag{4.10}$$

The *False Negative Rate (FNR)* is the percentage of the anomalies that are incorrectly predicted as normal cases (FN) over all the actual anomalies.

$$FNR = \frac{FN}{FN + TP} \tag{4.11}$$

The *F1-score* is the harmonic average of the precision and recall.

$$F1 - score = 2 \cdot \frac{precision \cdot recall}{precision + recal} \tag{4.12}$$

The above-mentioned preliminaries serve as a base for the proposed solution for the insider threat detection, presented in the following sections.

## 4.3 Proposed solution

Our approach is based on the design and monitoring of novel features from enterprise data that are reflective of insider threat behavior, and subsequent analysis of these data to identify inconsistent, statistically rare behavior that can be indicative of insider threat activity.

In Section 4.1, we presented the problem statement, denoted our assumptions and discussed the challenges to be faced. As a result of this analysis, this section details the proposed solution, with the following characteristics.

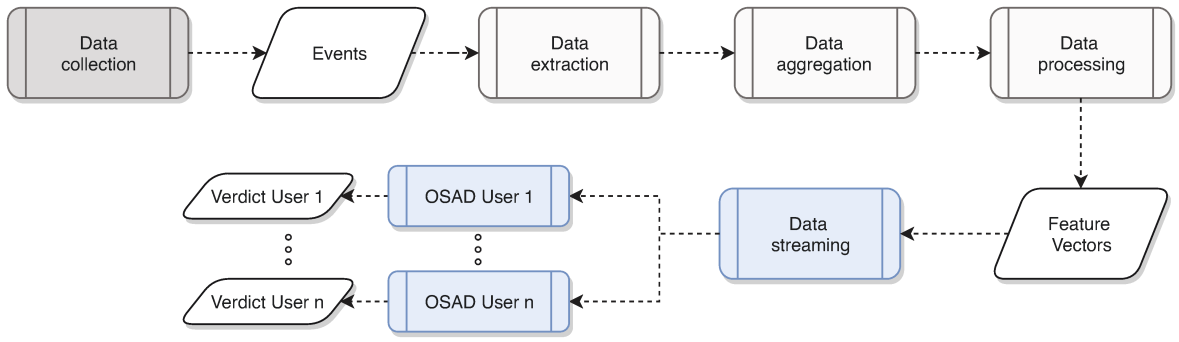



Figure 4.3 – General workflow of the three main modules of the Anomaly-based Detection Framework

**Clustering technique** As we have mentioned, threats are rare events; therefore we can state that in a learning clustering process, more dense clusters will denote the normal user behavior. Novel detection methods rely on using distance or density information to detect previously unseen data distribution patterns. These methods are capable of identifying uncertain suspicious samples which need further evaluation, and they define an additional "unknown" class label to indicate that these samples do not fit the existing view of the data.

**Unsupervised algorithm** Labeling is a time consuming and expensive activity, which often requires human intervention. The scale of modern-day machine learning applications and the volume of requests, makes labeling a luxury which is not affordable and is impractical. There is, therefore, a need to reduce the dependence on labeled samples, in the stream classification process. This chapter develops a reliable reactive approach to deal with unlabeled streaming data; therefore it is best-suited to work with an unsupervised learning algorithm.

**Employee's activities into feature vectors** Machine learning-based approaches have been central to work on insider-threat detection. The actions that an employee takes over a period on a system (e.g., logging on/off, file accesses) can be modeled as a sequence. The sequences that are seen often or on a usual basis can then be considered as the users' *normal* behaviour. Observed actions which do not resemble those *normal* sequences can be regarded as *anomalous* behaviour, which then may indicate a potential insider threat or at least an event to be investigated.

**Contextual analysis** We adopt an incremental approach to data collection, analysis, and decision making in which different data are collected and analyzed for different individuals depending upon insider threat attributes determined by the model, such as their position. Some observations or derived indicators may require an immediate response if they indicate malicious actions. However, the most sophisticated insiders likely operate more subtly, hiding behaviors within "normal activities" to elude detection. One of the challenging tasks in detecting insiders is to accurately model user behavior, based on such sequential data.

**Online stream detection algorithm** An online analysis enhances a CSC's need to react inconveniently; therefore we provide verdicts. For this matter, we choose to work with a detection framework that uses stream-data management. Consequently, the model is intended to make predictions on new data as it learns.

All five characteristics are assessed through a three-stage methodology. As depicted in Figure 4.3 and in more detail in the Section 5, our framework is constituted by:

1. Data collection of the employee's activities, through the monitoring of the cloud environment, mainly done through network's DPI/Deep Flow Inspection (DFI), and active monitoring modules.

2. Analysis of all the collected data, by means of data extraction, aggregation and processing through feature selection (Section 4.3.1).

3. Detection of all relevant features that serve as input for the OSAD module. This module relies on the clustering of the sequences and contextual information into groups, using the Euclidean metric as the distance measure.

4. Analysis of the employee-based clusters, where we identify a certain percentage of the outliers as the *anomalous* events.

5. Derivation of an anomaly score and perform an identification of where and why the event deviates from *normal* behavior.

In order to better feed our detection algorithm, an analysis of the extracted features is considered. This is part of the data processing phase from Figure 4.3 and it involves two steps (the following Section 4.3.1 and Section 4.3.2), for both, network and cloud collected data. The first step relies on visualizing relation between features, and also describing the variability and importance of these features information-wise. The second step concerns the feature scaling in order to better feed our algorithm.

### 4.3.1 Feature analysis

As a result, the monitored data are presented in the Table 4.3[2] and Table 4.4.

With respect to the relation between features, since our features dimensionality is not extremely high, we observe their correlation and variability. For that, the relationship between all features are depicted in Figure 4.4.

We monitored a total of 19 IaaS-based features related to the CSC's network, server instances and the images (e.g., Counts for server's resize, paused, hard reboot). Table 4.3 shows only the non-zero final features.

**IaaS-based feature set** descriptive statistics can be found in Table 4.3 and the correspondent pair plot in Figure A.1 in Annex A. From these two, we observe that only `Server Total Mean`, `Server Active Mean`, `Server Other Mean` and `Server Reboot Mean` are non redundant, in contrary of the rest (visible in Table 5.4), which always provide the same information. This is rather obvious given the fact that our simulated actions, in the scope of this thesis, did not consider any of the other monitored statuses towards the OpenStack implementation.

Nevertheless, given the fact our framework proposes the incremental addition of future actions, we implemented the whole possibility of the monitoring feature set. For example, if there is a new action that corresponds to suspend a VM or re-size a VM's RAM parameters, the features of `Server Suspended Mean` and `Server Resize Mean` would become relevant, respectively.

[2]The IaaS description details can be found in [99].

| Feature | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|
| Image Total Mean | 3.028E+00 | 1.746E-01 | 3.000E+00 | 3.000E+00 | 3.000E+00 | 3.000E+00 | 4.500E+00 |
| Image Total 25q | 3.025E+00 | 1.613E-01 | 3.000E+00 | 3.000E+00 | 3.000E+00 | 3.000E+00 | 4.250E+00 |
| Server Shutoff Mean | 1.622E-02 | 1.107E-01 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 1.000E+00 |
| Server Reboot Mean | 4.481E-01 | 3.853E-01 | 0.000E+00 | 1.250E-01 | 3.333E-01 | 1.000E+00 | 1.000E+00 |
| Server Total Mean | 1.145E+00 | 3.747E-01 | 0.000E+00 | 1.000E+00 | 1.000E+00 | 1.000E+00 | 2.000E+00 |
| Server Active Mean | 6.760E-01 | 3.888E-01 | 0.000E+00 | 5.000E-01 | 8.000E-01 | 1.000E+00 | 1.000E+00 |
| Server Delete Mean | 4.879E-03 | 6.759E-02 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 1.000E+00 |
| Server Password Mean | 1.884E-02 | 1.360E-01 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 1.000E+00 |
| Server Stop Mean | 1.636E-02 | 1.180E-01 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 1.000E+00 |
| Server Snapshot Mean | 2.041E-02 | 1.414E-01 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 1.000E+00 |
| Memory Mb Usage Var | 5.121E-05 | 1.899E-03 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 1.167E-06 | 1.364E-01 |
| Local Gb Usage Var | 5.093E-10 | 1.888E-08 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 1.161E-11 | 1.357E-06 |
| Server Total Var | 1.647E-03 | 2.011E-02 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 2.500E-01 |
| Server Total 75q | 1.146E+00 | 3.730E-01 | 0.000E+00 | 1.000E+00 | 1.000E+00 | 1.000E+00 | 2.000E+00 |
| Server Total 25q | 1.143E+00 | 3.772E-01 | 0.000E+00 | 1.000E+00 | 1.000E+00 | 1.000E+00 | 2.000E+00 |
| Security Rules Mod | 2.408E-02 | 1.533E-01 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 1.000E+00 |
| Server Severity | 1.456E+00 | 1.829E+00 | 0.000E+00 | 5.000E-01 | 1.000E+00 | 2.000E+00 | 1.800E+01 |
| Image Severity | 9.138E-02 | 5.156E-01 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 3.000E+00 |
| Network Severity | 1.685E-01 | 1.073E+00 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 7.000E+00 |

Table 4.3 – Descriptive statistics for IaaS-features for generated dataset

Other important fact while analyzing these features, is that we may be missing their correct values by the monitoring sampling time. This is the case, for example, of `Image Delete Mean`, where even though it is documented as a possible state, it is very immediate, therefore can be only captured in logs. To overcome this issue for this feature and others, we have used aggregated features that measure the variability of the total amount of servers and images, making this information accessible for the detection engine (more details regarding its calculation can be found in the next chapter).

**Network-based feature set** descriptive statistics can be found in Table 4.4 and their correlation plot in Figure A.2 in Annex A. From these two, we observe that some of the features show a relatively linear relation (e.g., `Total Packets` and `Total Bytes`) while for others it is more evident (e.g, `Total Packets` and HyperText Transfer Protocol (HTTP)'s content length; `Download Payload Bytes` and `Total Bytes`). To complement this information, Figure 4.4 depicts the correlation between all collected features. The reader can see that there are positive and negative correlations between some features.

Within the positive correlations we have the tuple `Download (DL) Data Vol` and `Total Bytes`. Thus, most of the times the user's activities rely on a download procedure, rather than an upload (hence the lower correlation with `Upload (UL) Data Vol`). This is fairly evident since the content length for a POST HTTP query is set to 0 while waiting for a HTTP response [42].

Also, for the relation between `Total Bytes` and `Session Time`, we observe there are two behaviors in the traffic: we see that there are high values of session time (in seconds) when `Total Bytes` has both low and high values. The reason may be due to the fact that employees may perform small sets of activities dealing with heavy loads (e.g., downloading a file), while also executing large sets of non-byte-demanding actions (e.g., editing the details of a VM) within a session. These two possible practices, of course, weaken the correlation (observable in Figure 4.4) making it a hard task to analyze them in an unsupervised manner.

Consequently, after this analysis, these two features are kept as relevant information at the moment of detecting the examples mentioned above.

| Feature | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|
| Total Bytes | 5.655E+08 | 3.561E+09 | 3.484E+03 | 3.848E+03 | 1.338E+04 | 1.906E+04 | 2.300E+10 |
| Total Pkts | 1.466E+05 | 9.232E+05 | 4.000E+01 | 4.400E+01 | 8.800E+01 | 8.800E+01 | 6.492E+06 |
| response Time | 1.531E+03 | 9.472E+02 | 5.906E+01 | 4.693E+02 | 1.592E+03 | 2.216E+03 | 6.347E+03 |
| UL Data Vol | 2.827E+08 | 1.780E+09 | 1.742E+03 | 1.924E+03 | 6.690E+03 | 9.532E+03 | 1.150E+10 |
| DL Data Vol | 2.827E+08 | 1.780E+09 | 1.742E+03 | 1.924E+03 | 6.690E+03 | 9.532E+03 | 1.150E+10 |
| Session Time | 1.515E+01 | 3.304E+01 | 3.614E-02 | 1.128E-01 | 3.171E-01 | 3.528E+01 | 3.403E+02 |
| Post Method Count | 2.997E+00 | 2.007E+00 | 0.000E+00 | 0.000E+00 | 4.000E+00 | 4.000E+00 | 1.200E+01 |
| Get Method Count | 5.623E+00 | 1.119E+01 | 0.000E+00 | 0.000E+00 | 4.000E+00 | 8.000E+00 | 8.400E+01 |
| Delete Method Count | 8.709E-02 | 6.008E-01 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 8.000E+00 |
| Transaction Count | 8.757E+00 | 1.110E+01 | 4.000E+00 | 4.000E+00 | 8.000E+00 | 8.000E+00 | 9.200E+01 |
| Interaction Time | 1.566E+00 | 9.288E-01 | 1.446E-01 | 4.885E-01 | 1.600E+00 | 2.233E+00 | 6.347E+00 |
| Application Type | 9.792E-01 | 1.428E-01 | 0.000E+00 | 1.000E+00 | 1.000E+00 | 1.000E+00 | 1.000E+00 |
| Octet Stream Type | 2.460E-02 | 1.549E-01 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 1.000E+00 |
| HTTP 200 | 5.393E+00 | 1.114E+01 | 0.000E+00 | 0.000E+00 | 4.000E+00 | 4.000E+00 | 8.400E+01 |
| HTTP 202 | 2.724E+00 | 1.907E+00 | 0.000E+00 | 0.000E+00 | 4.000E+00 | 4.000E+00 | 8.000E+00 |
| HTTP 204 | 8.458E-02 | 5.755E-01 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 4.000E+00 |
| HTTP 401 | 9.882E-02 | 6.209E-01 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 0.000E+00 | 4.000E+00 |

Table 4.4 – Network-based features descriptive statistics for dataset

On the other hand, `UL Data Vol` is somewhat correlated with the `Session Time`. We infer that this may be due to sessions that correspond to long list of actions, that are not bytes-demanding and/or that may take more time (e.g., an employee performing a list of activities in the search of a file).

In conclusion, by means of no over-fitting the further clustering steps, we choose to reduce our feature set by eliminating `Total Pkts`, since the same information can be deducted in `Total Bytes`.

**User-based feature set** can be viewed in Figure 4.4. As previously mentioned, differently from the data processing task based on network and IaaS-based feature vectors, the users' actions are represented by letters in the form of a list. However, this list does not have explicit features. For this purpose, from both training and testing datasets, we obtain the list of actions for each of the employee's simulated sessions. For every action (defined by letter) there is a correspondent letter in the following alphabet: [“D”, “L”, “P”, “S”, “R”, “T”, “Z”, “X”], as described in the previous Chapter 3; the implementation is explained in Chapter 5.

Experimental work has been carried out to determine the most suitable representation for a discrete sequence of actions. These results have been organized concerning the treatment of the strings in numerical in Table A.1 in Annex A.

From the corresponding experimental results detailed in Table A.1 in Annex A, we decided to use the alphabet representation of the employee's actions. By use of the sequence treatment presented in the preliminaries, we observed the relationship of these actions in relation with network and IaaS-based feature sets. This relationship is presented in Figure 4.4, where several inferences of the monitored data can be made.

The representation of the employee's actions is based in an $n$-gram, transforming strings into numerical vectors. As an example, for $n$-grams of length $n = 4$ , if the list of actions is [“L”, “L”, “R”, “L”], and “L” and “R” are the two existing actions in the alphabet, the resultant $n$-gram for that example will be the vector (3,1) if the order of representation for the alphabet is “L”, “R”.

Thus, for every action present in a sequence, the correlation plot in Figure 4.4 will show the relationship with the rest of the attributes. In other words, it represents the relationship with the rest of the features, whenever present.

At the same time, we cannot say much about the time-related features (e.g., `Response Time`, `Session Time`, `Interaction Time`) because these may not categorize the presence of these actions for different length sequences of actions.

Nevertheless, we can still infer that the action is "L" (i.e., action that is equivalent to listing the VM's details), for instance, when present, the feature `Total Bytes` and `UL Data Volume` are negatively correlated with it, while the response time is positively correlated. This characteristic indicates that this action is non-byte demanding, but it is embedded in long sessions or sequences of actions.

Additionally, we may see that the "D" (i.e., download image) action is in fact clearly correlated with the feature `DL Data Vol` and resembles also some relationship with the `Transaction Count`.

Also, we see that for the "T" (i.e., terminate instance) there is a weak correlation with our aggregated feature `Server Severity`.

Lastly, we observe that the latter is also weakly correlated with the "S" (i.e., stop instance) action, which both consist of security threats regarding service availability.

In conclusion, all these complex relationships motivate the use of `Server Severity`, `Image Severity` and `Network Severity` as contextual features, in order to help decrease the number of false rate positives in our detection algorithm.

### 4.3.2 Dynamic feature transformation process

The features selected in the precedent section rely on both numeric and symbolic or categorical values.

On the one hand, for the numeric features we proceed with a data standardization, which re-scales the input vectors to have a mean of 0 and standard deviation of 1 (unit variance). This relies on the fact that we have unbounded attributes (e.g., `Session Time`), where *anomalous* values could be possibly high. In this case, standardization (over normalization) is more useful, getting reasonable transformed values for *normal* values (otherwise the *normal* values will be "squished" by the outliers extreme values; for example, for values [0, 1, 2, 1E100], the normalization would result in [0.0, 0.0, 0.0, 1.0]).

Another reason to perform this feature scaling phase is that our dataset (composed by $m$ collected samples by the selected $n$ features) has features with different scales. For example, while `Server Active Mean` belongs to the interval $[0, 1]$ we have `Total Bytes` which can go from low values to as high as the HTTP communication is intended to be (e.g., a 1GB download). For that concern, most of machine learning algorithms are not scale-invariant (e.g., for ANN these different scales may affect the weight of the neurons passing through).

Hence, for a set of $m$ vectors $\mathcal{DS} = \{x_1, \ldots, x_m\}$, where each vector $x_i$ is $x_i = (x_i^1, \ldots, x_i^n)$ given by $n$ features, $\mu_n$ and $\sigma_n$ are the mean and standard deviation for the feature $n$. The standardization value is given by the Equation 4.13.

$$z_i = \frac{x_i^n - \mu_n}{\sigma_n} \tag{4.13}$$

One can see that subtracting the mean will only shift the feature distribution left (or right, if the mean is negative). Dividing by the standard deviation will only stretch the histogram horizontally and vertically. Neither operation will change the shape of the distribution.

In our proposed solution, we consider a clustering approach, by means of calculating the Euclidean distance between arriving data vectors. In this case, the values of the feature dimensions

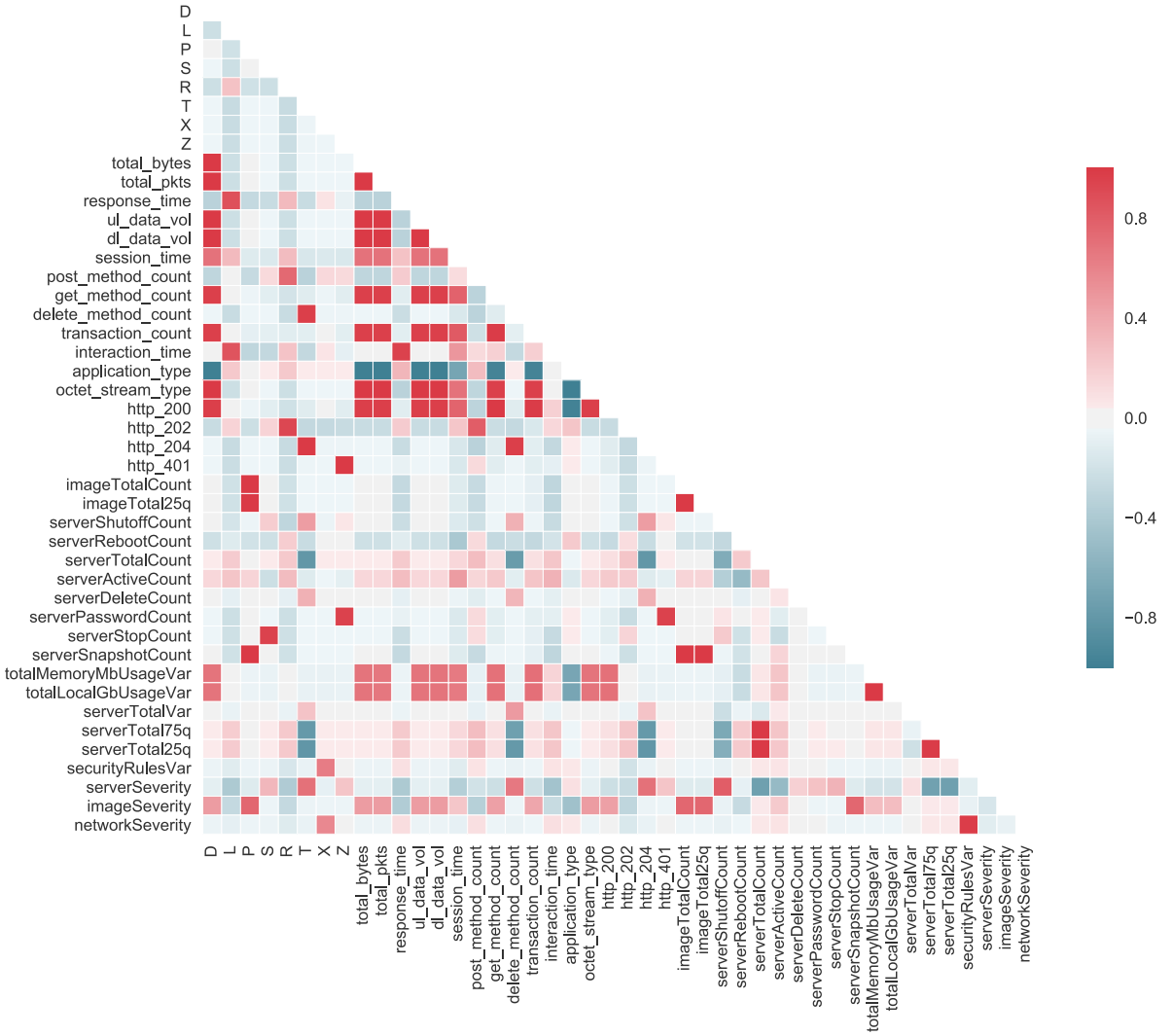



Figure 4.4 – Correlation plot for all features

that range between 0 and 1 may become uninformative and the algorithm would essentially rely on the single dimension whose values are substantially larger.

On the other hand, the categorical features are treated just as it is, in virtue of tagging each feature vector with *known* contextual information in the detection process.

Aside from these categorical features, by means of comparing our solution with different subset of features, we made use of the different employee's simulated actions, also collected from the same monitoring process, following the mentioned technique in Section 4.2.

As our solution adopts an offline and online phase, we note the importance of the offline training dataset size, since it will allow to fit the standardization scales for each feature more robustly. For the online phase, since our case involves an online data-stream, a continuous standardization of the data is required. This means whenever a new feature vector arrives, it will be treated accordingly by this module and then applied to the main algorithm.

### 4.3.3 Online Streaming Anomaly Detection Algorithm (OSAD)

In this section, we introduce the OSAD and describe some of its novel contributions within the detection of the insider threat. The structure of the following work concerns:

- The internal details of the proposed algorithm through novel contributions with respect to the original algorithm detailed in Section 4.2, in the application for the insider threat in Section 4.3.3.1;
- The detection methodology through the first learning phase in Section 4.3.3.2 along with the novel labeling proposal in Section 4.3.3.3.

First of all, the algorithm implements a data-stream approach, where the data are treated on-the-fly by the algorithm. As mentioned in the previous section, this stream is defined as a list of vectors $\mathcal{DS}$, namely $x_1, \ldots, x_m$, data streams arriving at time $t_1, \ldots, t_m$, where $x_i = (x_i^1, \ldots, x_i^n)$ is a vector in $\Re^n$ space.

At every time-step $t$ the model is represented by a graph $\mathcal{G}$ where each neuron represents a cluster, following the GNG baseline described in Section 4.2.3.

Our detection procedure functions in two phases: offline and online learning. At the first phase, the objective is to make the ANN learn the intrinsic relationships between a large dataset. This is based on the assumption that by capturing user activities in a medium-term, we are able to analyze the normal usage and overall behavior for each employee.

At the second phase, the objective is to detect and predict anomalies near real time. The proposed algorithm extends GNG and analyses new data for deciding if every incoming input belongs to an anomaly or to a normal behavior for the user. Both phases are explained in detail in the following sections.

#### 4.3.3.1 OSAD and other models' comparison

The Algorithm 5 in Section 4.2 explained how the original GNG can be resumed in 4 dominant procedures: (i) **Learning**, (ii) **Insertion**, (iii) **Deletion**, and (iv) **Stopping**. The OSAD algorithm proposes to go beyond the original methodology proposed in [43], [44], [126] by adapting these functionalities to an evolving network to properly detect insider threat anomalies.

**Learning** As said in Section 4.2.3, the learning rate determines the extent to which the winner and the neighbors of the winner are adapted towards the input signal.

The original GNG in Algorithm 5 learns through fixed parameters $\alpha_1$ and $\alpha_2$ for $bmu_1$ and its neighbors, respectively, as shown in Equation 4.1. This is a disadvantage that makes the network very sensitive to new incoming data. In other words, every time a neuron "wins", it will update its prototype by a factor of the absorbed input.

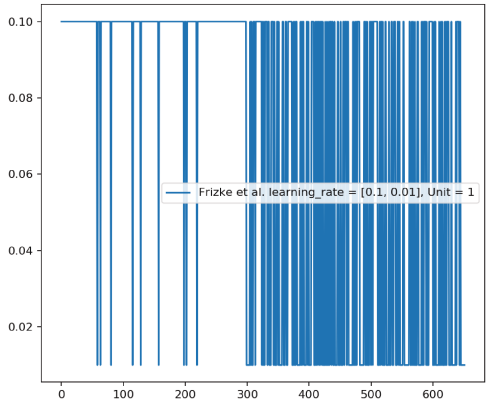
Consequently, since this learning is done always by the same factor and independently of the network evolving characteristics, it outcomes that every neuron will be much more susceptible to the input pattern they lastly won, making this solution more prone to errors in the case the neuron wins an input with a high distance from its prototype.

Additionally, the two detailed adaptive propositions of [44], [126] make use of the number of times the neuron has won, by means of decreasing its learning once it has absorbed many inputs. This means the more it wins, the less it needs to keep "learning". This can also be seen in Figure 4.5, which depicts the evolution of the learning rate parameter for the most winning

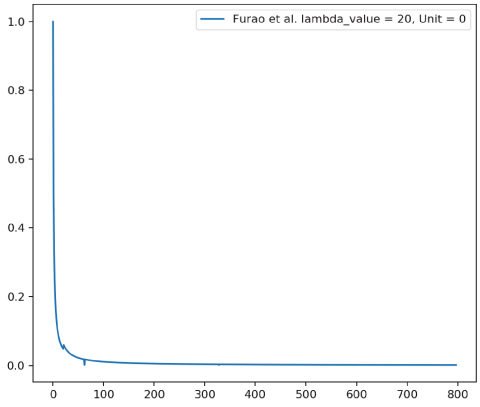
neuron for the three mentioned approaches. For Figure 4.5b, the curve is $1/H_{bmu_1}(t)$, as the neuron increases its winning, it becomes more stable, i.e., its prototype is more specific to the particular set of won inputs. This is an advantage because it may result in a graph that is able to cluster more specifically the many different behavior variations of an employee.

In order to have a more efficient learning function for the insider threat, experimental comparisons of the three approaches were performed and are detailed in Chapter 6, where all three are compared for the four presented dataset scenarios in this work, as well as for different employee's profiles.

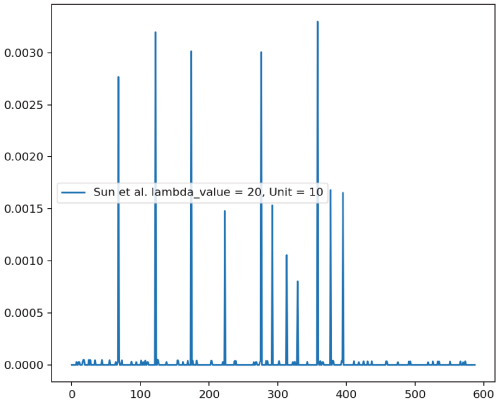
Moreover, in contrast to the original fix approach in Figure 4.5a, the results of these experiments in Section 6.3.1 prove that the approach in [44] favors a higher precision and lower FPR, along with higher silhouette scores. The second adaptive approach presented in [126] and depicted in Figure 4.5c showed fluctuations, given by the decay factor of the exponential function of Equation 4.3.




(a) Fritzke [43] with fix learning rate $\alpha_1 = 0.1$ and $\alpha_2 = 0.01$



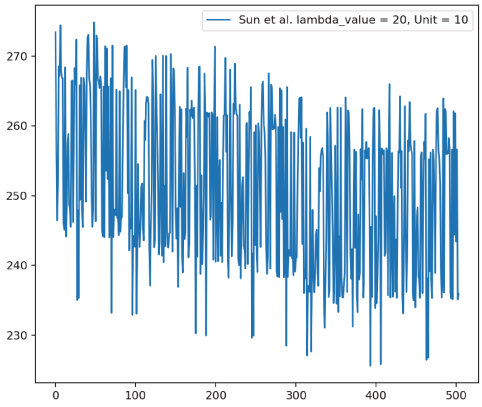

(b) Furao and Hasegawa [44] learning rate (Equation 4.2)



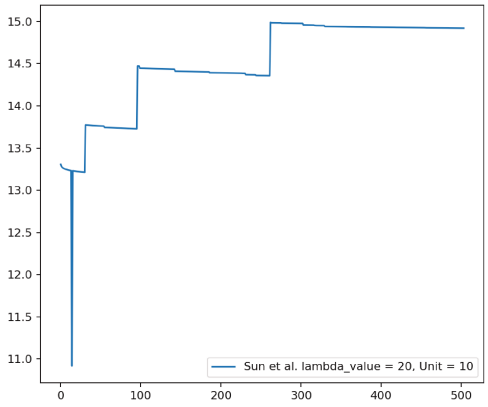

(c) Sun et al. [126] learning rate (Equation 4.3)

Figure 4.5 – Different learning rates for highest winning neuron toward 3 models with $\lambda = 20$

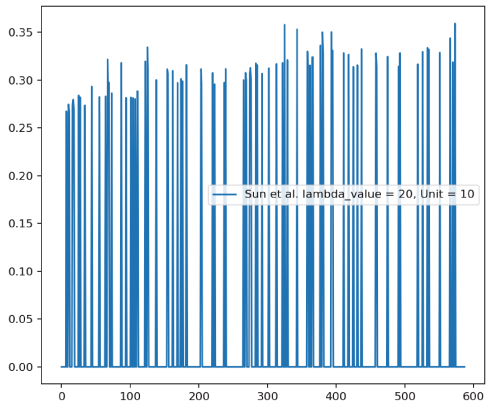
The threshold parameter of Equation 4.3 was formerly used in the work of [44], after a long period (not specified in their paper) of training. The authors intended to use this as a similarity threshold to classify the graphs in the neurons, distance-wise, once the algorithm had learned and did not present major changes. Also, authors of [126] utilized this parameter at every time step to calculate how the winning neuron is spatially situated with respect to its neighbors. Although this may be an interesting spatial attribute to consider, it may be too sensitive to the natural changing behavior of the incoming data, as well as the $\lambda$ insert parameter, as shown in Figure 4.6.




(a) Numerator of the Equation 4.3 correspondent to the Euclidean distance between the input vector and the neuron's prototype




(b) Denominator of the Equation 4.3 correspondent to the maximum distance between the neuron and all its neighbors




(c) Exponential function of the Equation 4.3

Figure 4.6 – Adaptive learning rate of [126] (Equation 4.3)

To better understand the fluctuations from Figure 4.5c, a detailed view of each of the components of Equation 4.3 is shown in Figure 4.6. The images represent the behavior of the neuron that won more times in the graph.

On the one hand, from Figure 4.6a, one can see as the neuron learns, its starts decreasing its

distance. Every fluctuation indicates the neuron won among the others, however the absorbed input was rather different than its prototype. This could also be read as the absorbed input was a "surprise" for the neuron. On the other hand, from Figure 4.6b a reader can see there are not many differences in the recorded time lapse. Moreover, every step in the figure means that the region of the neuron and its neighbors is increasing spatially.

Therefore, visible peaks in Figure 4.5c are due to all the presented factors. Also, the mentioned $\lambda$ also controls the neuron's insertion, which can also modify the neighborhood of the $bmu_1$. Hence, not only the adaptiveness of the learning procedure is important, but also the insertion rate.

In terms of insider threat behaviors, this could be sensitive to the heterogeneity of the employee's actions towards the cloud assets.

As a result of this analysis, this work adopted the adaptive learning rate over time presented by [44] in Equation 4.2.

**Insertion** The original GNG of Algorithm 6 inserts neurons at an arbitrary fixed-rate (i.e., every $\lambda$ steps). This is a disadvantage given the nature of the network, which may evolve as new behaviors happen. Hence, it is necessary to apply a different approach as in [44], [126] through a different criterion for neuron insertion. Inspired in the latter ideas, the proposed algorithm runs with the presented adaptive approach in [44], [126], inserting neurons only when the input vector's distance to the BMU is bigger than an adaptive threshold.

Additionally, this work extends the previous models by inserting neurons via a domain knowledge contextual data, derived in Chapter 5. The experimental comparison by means of using the proposed extension, with the different insider threat scenarios is detailed in the experimental results in Chapter 6.

**Algorithm 7** OSAD Neuron Insertion

**Input:** $x_i$ and model params $\mathcal{M} = (\lambda, \alpha_1, \alpha_2)$, $T_{bmu_1}$, $w_{bmu_1}$
**Output:** Neurons $\mathcal{N} = \{u_1, \ldots, u_n\}$ and prototypes $\mathcal{W} = \{w_1, \ldots, w_n\}$

**if** $x > T_{bmu_1}$ **or** ContextualSeverityCriterion **then**
  **if** CheckInReservoir($x_i, w_{bmu_1}$) **then**
    InsertFromReservoir
  **else**
    InsertOuterNeuron
  **end if**
  Continue
**end if**
**if** $i$-th x data is an integer multiple of $\lambda$ **then**
  InsertNeuron (Algorithm 6)
**end if**
Update the local accumulated number of wins $H_{bmu_1} \leftarrow H_{bmu_1} + 1$

Moreover, by means of reviewing the efficiency of the past approaches in comparison with the proposed in this work, Figure 4.7 shows how the network size changes for every model. For this particular experiment, training data consisted in 800 samples of 36 features each, feature set correspondent to the scenario 1 described in Section 6.2.1 from Chapter 6. The experiments

from Figure 4.7 correspond to 18 iterations over the same data. The insertion $\lambda$ step was 50, hence the step curves in Figure 4.7a.

**Algorithm 8** OSAD Contextual Severity Criterion

**Input:** $x_i$, prototypes $w(i)$ $\forall i \in$ {server, image, network} severity
**Output:** Boolean

1: $list_i \leftarrow w(i)$, $\forall i \in$ {server, image, network}
2: $T_i \leftarrow mean(list_i) + std(list_i)$, $\forall i \in$ {server, image, network}
3: boolean_n $\leftarrow$ **if** $x(server\_severity) > T_{server}$
4: boolean_i $\leftarrow$ **if** $x(image\_severity) > T_{image}$
5: boolean_s $\leftarrow$ **if** $x(network\_severity) > T_{network}$
6: Return boolean_n **or** boolean_i **or** boolean_s

From Figure 4.7 a reader can see that overall the proposed OSAD models have a low global error (Figure 4.7c), a steady insertion/deletion rate (Figure 4.7a) while keeping its neurons with absorbed inputs.

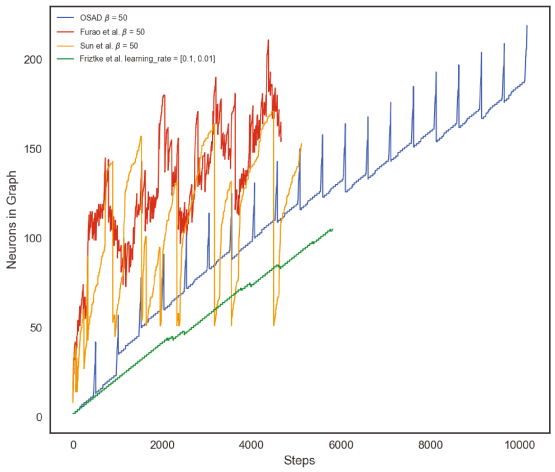
It can also be seen that for the utilized dataset, the author of [43] (green curves in Figure 4.7) have the lowest number of neurons in the graph $\mathcal{G}$. This is the result of an insertion rate that is not sufficiently efficient, given the fact that at the end of every iteration, more than half of the created neurons are deleted because they never won any input. This is also visible in Figure 4.7b where the number of neurons with absorbed inputs is rather stable. This means that the tendency is that incoming inputs are absorbed by the same set of neurons. In consequence, for different input behaviors, the error of the neuron will grow, as seen in Figure 4.7c.

**Deletion** The original GNG of Algorithm 6 eliminates edges when the age value exceeds the $age_{max}$. This results in some isolated neurons to be deleted. An extension of this work was provided by authors in [44] who added a parameter for every neuron's winning frequency or density of absorbed inputs, in order to delete isolated neurons under a frequency threshold parameter. On the other hand, authors in [126] also included the distance from these neurons towards their neighbors.

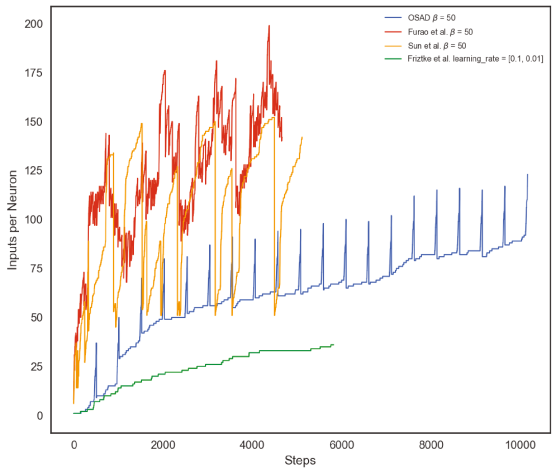
Although in all previous works deletion has the main functionality of dismissing obsolete neurons while prevailing the scalability of the network, this is a disadvantage for an anomaly detection case scenario. Even though it allows the network to be highly dynamic in terms of representing new evolving behaviors, it may discard anomalies as they are rare events. Hence, it is necessary to apply a different approach through a different criterion for neuron deletion.

The proposed solution in the present work, concerns two differences with respect to previous works. First, the deletion procedure is executed at the end of every iteration (i.e., a full run of data feed to the algorithm), therefore the adaptive threshold is only calculated at this step. This benefits from allowing the graph to adapt for a longer period of time, and make better use of newly inserted neurons that have not won any incoming vector. This advantage is also seen in Figure 4.8 where the proposed algorithm has a lower deletion rate, hence it keeps more data in the graph, while keeping good global error values, such as in Figure 4.7c, with respect to the other models.

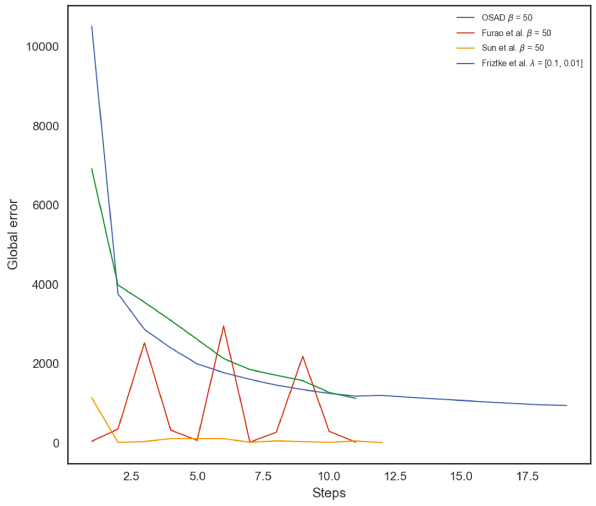
Second, the obsolete neurons and its neighbors are kept in memory, by use of a *reservoir*, which is a generic concept utilized in previous stream clustering approaches (e.g., [50], [147]).




(a) Number of neurons in the graph




(b) Number of neurons with absorbed inputs




(c) Global error for all the neurons in each graph

Figure 4.7 – Neuron insertion results of OSAD algorithm and studied models with insert rate of $\lambda = 50$ for every iteration step

This is only entitled for neurons labeled as *treacherous*. The intuition behind this strategy is to keep in the *reservoir* the neurons that statistically have shown to be out of the normal behavior for each severity feature.

This assures that for a predefined period of time, the data absorbed and therefore the spatial structure by these neurons is not lost. The details of the procedure are described in Algorithm 7. Consequently, whenever a new input arrives, it will be firstly analyzed by the neurons in the *reservoir*.

Moreover, the obsolete concept differs in the compared algorithms and this criterion also affects the overall behavior of the learning process. Whether to delete (or send to the *reservoir*) isolated neurons or neurons with less than a number of neighbors was also a matter of study. This matter also depends on the moment planned to do the deletion. For instance, if the deletion is meant to be checked after every insert (such as [44], [126]) or after every adaption (such as

[43]) the result will depend also on the amount of data the network has absorbed so far and how the connections have been made for the *normal* and *anomalous* data.

On the one hand, if the deletion is made at the end of every iteration (for a fixed number of data inputs), the procedure will consider a much more trained network, therefore will eliminate poorly connected neurons after that longer time. On the other hand, this procedure may result in the creation of a higher number of isolated-never-won neurons. Therefore, the moment of executing this functionality is not a trivial task as it depends on the data.

To tackle this issue with a conservative approach, the proposed algorithm deletes neurons with less than two neighbors, after a predefined number of read inputs, namely iteration. At that point it first considers those neurons that have no absorbed inputs. Later, and with the purpose of preserving the possible *treacherous* neurons, it stores those in the *reservoir*.

**Algorithm 9** OSAD Neuron Deletion

**Input:** $H$, $\mathcal{G}$, $w = (w(image), w(server), w(network))$
**Output: Reservoir**

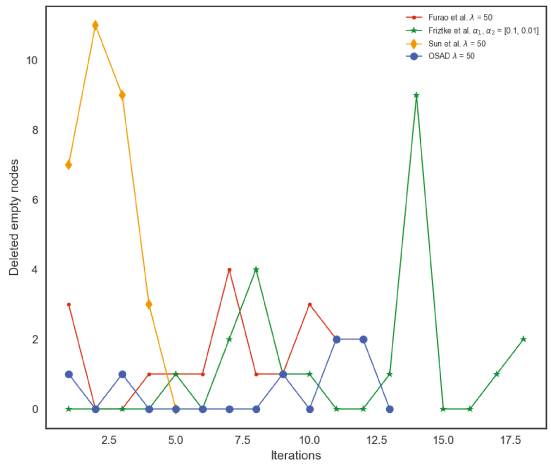
```
1: Reservoir ← {}
2: T_h ← mean(H)
3: T_server, T_image, T_network ← GetSeverityThresholds(w)
4: neurons ← GetNeuronsWithLessThanTwoNeighbors(G)
5: for all n ∈ neurons do
6:     if H_n is T_h then
7:         G ← DeleteNeuron(n)
8:         Continue
9:     end if
10:    server_n, image_n, network_n ← GetNeuronSeverity(n)
11:    if server_n > T_server or image_n > T_image or network_n >
       T_network then
12:        Reservoir[n] ← w_n
13:        G ← DeleteNeuron(n)
14:    end if
15: end for
```

All adaptive thresholds, namely $T_h$, $T_{server}$, $T_{image}$, $T_{network}$ proposed in the Algorithm 9 are calculated at the moment of finishing an iteration. Therefore, they are updated through the calculation of the current state of the graph $\mathcal{G}$ for the set of number of hits $H$, server severity prototypes, image severity prototypes and network severity prototypes, respectively.

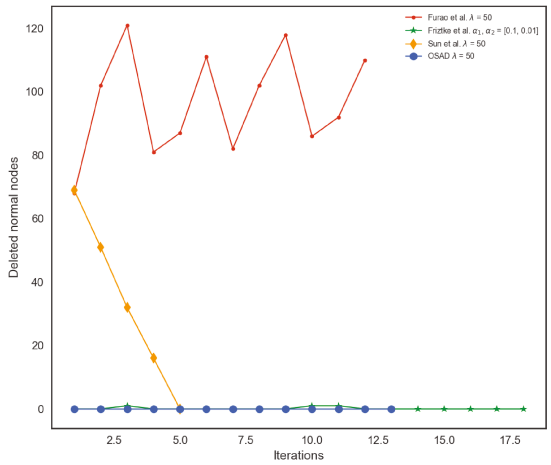
For the latter, the severity calculation strategy for every soon-to-be-deleted neuron is described in Algorithm 8. Under the assumption that most of the three severities behave under certain regular values (mean), the strategy considers all *reservoir* candidates which $w_i$ is higher than the mean plus one standard deviation.

For greater detail, Figure 4.8 depicts the distribution of the deleted neurons for the proposed algorithm, in comparison with the other models. For this purpose, labels have been included in the analysis to verify the *ground-truth* behavior of the procedure.

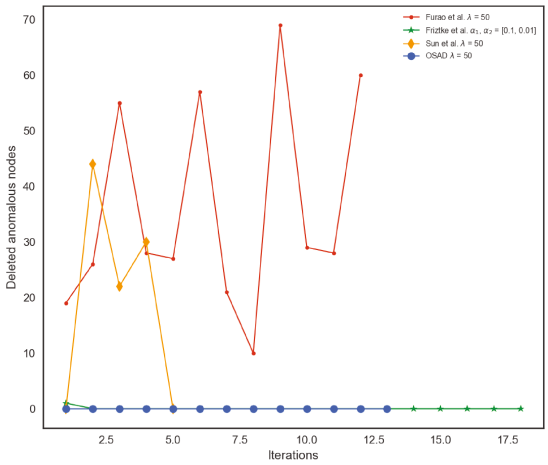
A deeper analysis was carried for the remaining deleted anomalous neurons in Figure 4.8c. These neurons consequently correspond to either neurons with few absorbed inputs or with a value for any of the severities which are not that critical. The first case relies on the fact that




(a) Number of *empty* neurons deleted in the graph




(b) Number of *normal* neurons deleted in the graph




(c) Number of *anomalous* neurons deleted in the graph

Figure 4.8 – Neuron deletion results of OSAD algorithm and studied models $\lambda = 50$

each neuron's prototype $w$ is updated by a fraction of the absorbed input through the previously mentioned Algorithm 5 (line 7). Therefore, the neuron with only few inputs learns through this iteration cycle and at the moment of the deletion phase, it may not reassemble in the exact spatial form to its absorbed inputs. The other finding relies on the fact that even if a neuron may have learned rapidly (i.e., low error) and may represent accurately the characteristics of its absorbed inputs, it may happen that the inherent severity characteristics of that collection of inputs may not be higher than the threshold $T_{server}$, $T_{image}$, $T_{network}$. An example of this case could be a *stop* VM action (in contrast to a *delete* VM action), which may have not been defined as highly critical by an expert, therefore, it will not be higher than the threshold defined in Algorithm 8.

***Reservoir*** By utilizing the contextual severity criteria in Algorithm 8, the proposed methodology benefits itself from the use of a *reservoir* for keeping neurons with certain *treacherous* characteristics. The workflow of this procedure is succinctly extracted from the algorithm's output and listed in the following example.

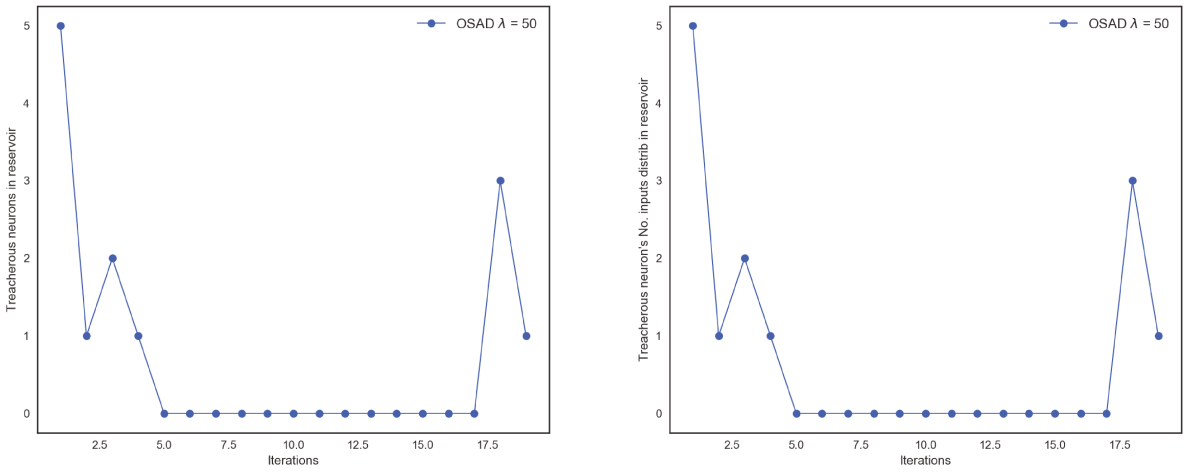
**Example of the algorithm's output for *reservoir* management.**

1. Iteration 1
   - Deleting neurons with less than 2 neighbors: neuron 2, neighbors: 26, (sequence label count: "T": 4). Winning hits less than thresh $21.56 < 100.12$
   - Sending neuron 2 to *reservoir* due to Severity Server: `True`, Image severity: `False`, Network severity: `False`
2. Iteration 2
   - Learning phase: Including neuron 2 from *reservoir*
   - Deleting neurons with less than 2 neighbors: neuron 2, neighbors: 26, (sequence label count: "T": 5). Hits less than thresh $23.56 < 96.83$
   - Sending neuron 2 to *reservoir* due to Severity Server: `True`, Image severity: `False`, Network severity: `False`
3. Iteration 3
   - Learning phase: Including neuron 2 from *reservoir*
   - Deleting neurons with less than 2 neighbors: neuron 2, neighbors: 26, (sequence label count: "T": 6). Hits less than thresh $25.56 < 100.74$
   - Sending neuron 2 to *reservoir* due to Severity Server: `True`, Image severity: `False`, Network severity: `False`
4. Iteration 4
   - Learning phase: Including neuron 2 from *reservoir*
   - Deleting neurons with less than 2 neighbors: neuron 2, neighbors: 26, (sequence label count: "T": 7). Hits less than thresh $27.56 < 98.53$
   - Sending neuron 2 to *reservoir* due to Severity Server: `True`, Image severity: `False`, Network severity: `False`

The overall behavior of the *reservoir* can be seen in Figure 4.9, which describes the evolution of the number of neurons and their absorbed inputs across the learning steps. From the illustrations it can be seen the values are the same for both, the number of neurons, and for the number of inputs for each neuron. This results shows that for this example, all the neurons within the *reservoir* contain only 1 absorbed input.

**Stopping** The original GNG of Algorithm 5 can use different stopping criteria such as the maximal number of neurons and the global error between two training iterations. Authors in [126] proposed a stopping criterion based on the silhouette scores' variation (defined in Section. 4.2.5) in time.

Although this metric may accurately depict the clustering quality of the algorithm, this fixed criterion has to be evaluated for the insider threat detection scenario. As also previously mentioned, the proposed algorithm works with a streaming approach; therefore the procedure can learn in a continuous manner, absorbing new inputs that could be part of new user behaviors.



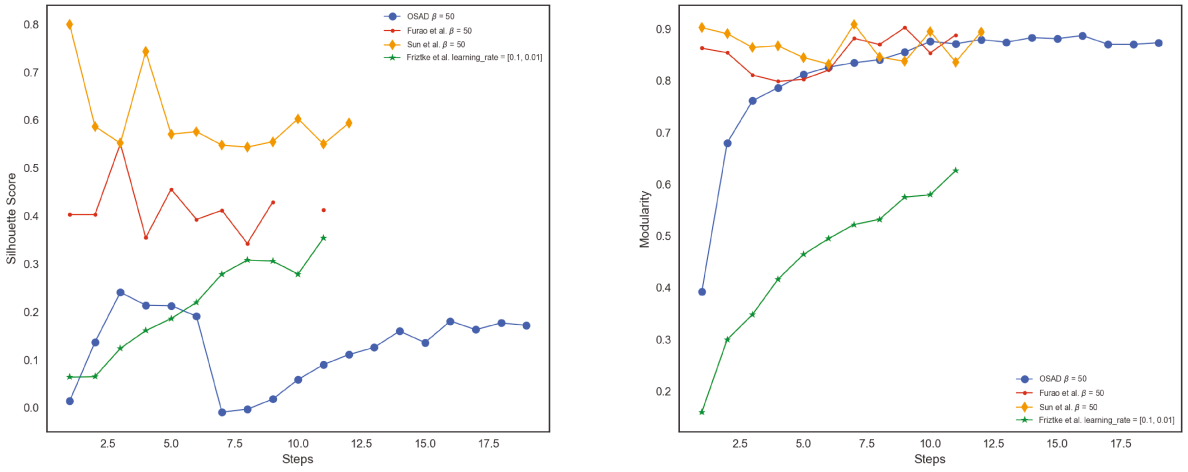

(a) Number of *treacherous* neurons (b) Number of inputs of all *treacherous* neurons

Figure 4.9 – *Reservoir* behavior for the OSAD algorithm

The above-mention factors encourage the idea of a no-stopping criterion unless the algorithm is meant to be trained and tested with the same data behaviors in time.

In order to evaluate the latter case, experimental results for a large number of iterations have been made, justifying the fact that for that given case, the algorithm reaches a stable state. The result can be seen in Figure 4.7b, which describes the number of neurons in the graph $\mathcal{G}$ through iteration steps, showing the tendency that existing neurons and not new neurons tend to absorb new incoming inputs.

The same tendency can be seen from Figure 4.10, where the graph modularity reaches stability after iteration 10 for the OSAD algorithm.




(a) Silhouette score (b) Modularity

Figure 4.10 – Internal metrics' results for proposed studied models with insert rate of $\lambda = 50$

The silhouette coefficient, on the other hand, relies on the prototypes of the neurons (i.e., the data characteristics of the inputs absorbed by them), and tests the quality of the intra-cluster

and inter-cluster similarity of them. Thus, of course, this coefficient is going to be heavily dependent on the number of neurons the algorithm removes or keeps in the graph. Such is the case of the algorithms of [126] and [44], which have higher silhouette scores. However, their deletion rate is higher than the OSAD algorithm.

#### 4.3.3.2 Phase 1.1: Offline training

Once the algorithm has integrated all data points into the neuron topology, the present work proposes to combine this technique with the use of *communities*. Although there is no universal definition of what a community is, most existing algorithms rely on the principle that neurons tend to be more connected within a community than across communities. This scheme is a simple, efficient and easy-to-implement method for identifying families of clusters in large networks.

**Algorithm 10** Normal Cluster Labeling

**Input:** Graph $\mathcal{G}$, $T_h$, Communities $\mathcal{C} = \{c_1, \ldots, c_n\}$
**Output:** Normal neurons $\mathcal{N} = \{n_1, \ldots, n_m\}$

1: $medoids \leftarrow$ GetMedoids($\mathcal{G}, \mathcal{C}$)
2: $\mathcal{N} \leftarrow \{\}$
3: $count\_per\_com \leftarrow$ GetAbsorbedInputsPerCommunity($\mathcal{C}$)
4: $\mathcal{NC} \xleftarrow{+}$ PercentileCriteria($count\_per\_com$)
5: $\mathcal{NC} \xleftarrow{+}$ MostWinningNeurons($T_h, C$)
6: $\mathcal{NC} \xleftarrow{+}$ NearestCommunities($\mathcal{NC}, medoids$)
7: $\mathcal{N} \leftarrow$ All neurons from $\mathcal{NC}$
8: **function** PercentileCriteria($count\_per\_com$)
9: **for all** community_counts $\in count\_per\_com$ **do**
10: **if** $(community\_count, community) \geq percentile(count\_per\_com)$ **then**
11: $\mathcal{NC} \xleftarrow{+} community$
12: **end if**
13: **end for**
14: **end function**
15: **function** MostWinningNeurons($T_h, C$)
16: **for all** $n \in$ GetNeuronsWithHitsBiggerThan($T_h$) **do**
17: $\mathcal{NC} \leftarrow$ community where $n$ belongs
18: **end for**
19: **end function**
20: **function** NearestCommunities($\mathcal{NC}, medoids$)
21: max_intra_cluster $\leftarrow max($GetCommunitiesMSE($\mathcal{NC}, medoids$)$)$
22: **for all** c $\in \mathcal{NC}$ **do**
23: **if** c $\leq$ max_intra_cluster **then**
24: $\mathcal{NC} \xleftarrow{+} c$
25: **end if**
26: **end for**
27: **end function**

Consequently, the partition of the graph $\mathcal{G}$ is computed, which maximizes the modularity

using the Louvain heuristics. The resultant partition is the neurons' grouping that ends with the highest modularity. This procedure allows the algorithm to obtain cluster "families" for further labeling.

#### 4.3.3.3 Phase 1.2: Cluster labeling propagation

We proposed a strategy for labeling the communities shown in Algorithm 10.

Through the new community structure for the graph, an analysis is made to every community and its belonging neurons. By means of detecting the employee's behavior through their clustered data, three labeling behavior categories have been defined:

- **Normal**: behavior related to the former assumptions of this work, described in Chapter 1.
- **Treacherous**: behavior that is related to the highest severity labels (i.e., network, image and server) and/or contextual anomalous levels (i.e., outside working hours).
- **Unknown**: behavior that has not been possible to classify in any of the previous two.

**Normal labeling** procedure is done by three criteria, namely PercentileCriteria (line 8) MostWinningNeurons (line 15), and NearestCommunities (line 20).

Firstly, under the second assumption in Section 4.1, the number of input vectors that fell under each community of neurons it is considered. Afterwards, the threshold correspondent to the percentile by which the distribution is equal to the mean plus one standard deviation, is calculated. A conservative criterion is therefore used by labeling as normal communities ($\mathcal{NC}$) those that have more inputs than this threshold.

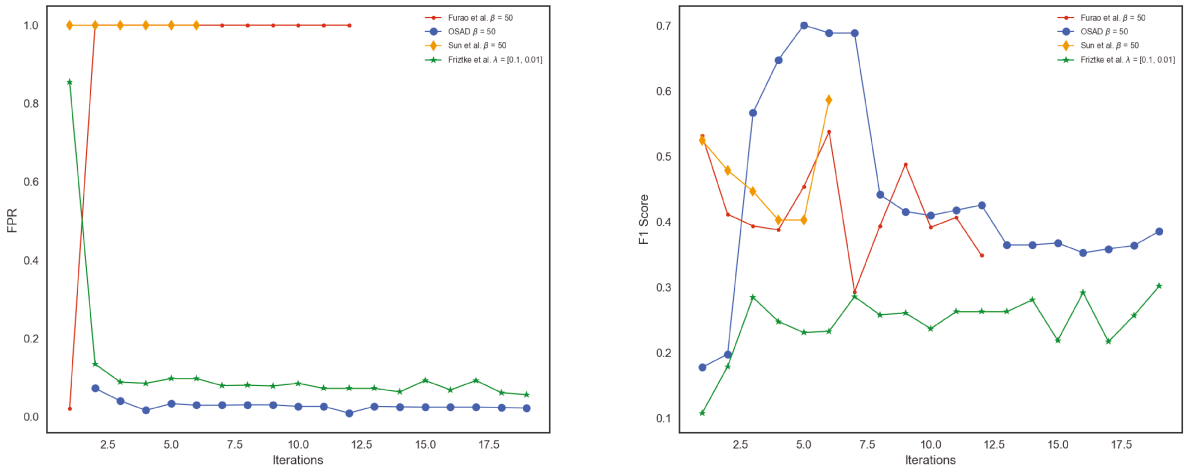
Consequently, the algorithm considers as $\mathcal{NC}$ only the clusters which have the number of inputs as high as that percentile of the cluster's input distribution. This value is set adaptively, and therefore, it depends on the historical data distribution of the absorbed inputs for all the neurons in the graph $\mathcal{G}$ after every iteration step.

Nevertheless, it could happen that our training data are not condensed into one big *normal* cluster with other smaller ones, but that is instead spread through multiple clusters with very short distances between them. Moreover, from experimental evaluations and mentioned insider threat scenarios (e.g., [17]) it was seen that there are many kinds of *normal* activities, giving small spread clusters that may not even be close to each other. In the context of cloud CSC employee's actions, this may be due to:

- Employee's actions varying broadly in time.
- Employee's actions not being executed in the same predefined manner.
- Employee's new actions currently gaining cluster density.

This possibility would result in the labeling of such *normal* clusters, as *unknown*. To overcome this case, if there is no clear bigger cluster, the second MostWinningNeurons proposed criterion is utilized, which labels as *normal*, the communities where the set of *highest-winning* neurons' reside.

The last criterion makes use of a second-level topology analysis, which aims at relating communities based on their inter-cluster distance. The intuition behind this procedure is to consider communities that are close to each other, to have the same label. Accordingly, NearestCommunities obtains the medoid for each community and calculates the distance between them.




(a) FPR (b) F1 score

Figure 4.11 – External evaluation metrics' results for proposed studied models with insert rate of $\lambda = 50$

Afterwards, each pair of communities' distance is compared against the average inter-cluster distance of the whole graph.

**Ttreacherous labeling** procedure is done by reason of using the contextual information, as seen in Algorithm 11.

Both algorithms label the same set of communities, therefore it could happen that a community is labeled both as *normal* and *treacherous*. In this case, the final label for this community is *unknown*. This might be the case of a community cluster with both, an amount of normal behaviors, as well as inputs with high severities.

**Algorithm 11** Treacherous Cluster Labeling

**Input:** Graph $\mathcal{G}$, Communities $\mathcal{C} = \{c_1, \ldots, c_n\}$
**Output:** Treacherous neurons $\mathcal{T} = \{t_1, \ldots, t_l\}$

1: treacherous neurons ← GetNeuronsWithMaxSeverity($\mathcal{G}$)
2: **for all** $n \in$ treacherous neurons **do**
3: $\mathcal{TC} \leftarrow$ community where $n$ resides
4: $\mathcal{T} \leftarrow$ all neurons of community where $n$ resides
5: **end for**

#### 4.3.3.4 Phase 2: Online streaming detection

The proposed framework ultimately aims to evaluate a continuous stream of events in an online manner. To achieve this goal every new incoming vector is absorbed by the trained/learned offline representation of the OSAD (i.e., phase 1). The Algorithm 12 outlines the proposed online stream version of the OSAD approach. In this version, unlike the original approach of GNG, the data are seen only once (i.e., stream), there is not an end of data arriving (i.e., online)

and the predictions with respect to possible anomalies are made at predefined times, near real time. The latter is done by determining the overall behavior status of the graph $\mathcal{G}$.

We also note the depiction of the precedent algorithms, denoted by OSADNeuronInsertion(·), OSADNeuronDeletion(·), NormalClusterLabeling(·) and TreacherousClusterLabeling(·), which consider for their calculation, each respective function arguments under the notation (·).

**Algorithm 12** Online Streaming Anomaly Detection (OSAD)

**Input:** $\mathcal{DS}$ and model params $\mathcal{M} = (\beta, \alpha_1, \alpha_2, \eta)$, $\mathcal{E}$, $\mathcal{W}$, $\mathcal{N}$
**Output:** Anomaly score $s_t$

1: **for all** x $\in \mathcal{DS}$ **do**
2: $bmu_1 \leftarrow \text{argmin}_{bmu_1 \in U} \|x - w_{bmu_1}\|$
3: $bmu_2 \leftarrow \text{argmin}_{bmu_2 \in U \setminus bmu_1} \|x - w_{bmu_2}\|$
4: **if** neuron insertion is necessary **then**
5: OSADNeuronInsertion(·) (Algorithm 7)
6: **else**
7: Modify the age of all edges emanating from $bmu_1$
8: Add the distance between $x$ and $bmu_1$ to error variable $error(bmu_1)$
9: **end if**
10: **if** $i$-th x step is multiple of $\eta$ **then**
11: OSADNeuronDeletion(·) (Algorithm 9)
12: **end if**
13: Decrease the error of all neurons: $\mathcal{E} \leftarrow \mathcal{E} - \beta * \mathcal{E}$
14: Unknown neurons $\mathcal{U} \leftarrow \mathcal{N}$
15: Normal neurons $\mathcal{L} \leftarrow$ NormalClusterLabeling(·) (Algorithm 10)
16: Treacherous neurons $\mathcal{T} \leftarrow$ TreacherousClusterLabeling(·) (Algorithm 11)
17: $s_t$ for *normal*, *treacherous* and *unknown* $\leftarrow \frac{length(\mathcal{L})}{N}, \frac{length(\mathcal{T})}{N}, \frac{length(\mathcal{U})}{N}$
18: **end for**

Moreover, the final prediction component (line 17) discussed is anomaly estimation. The model produces anomaly scores, which are used to rank each user's graph $\mathcal{G}$ giving a ratio for *treacherous* and/or *anomalies*, *normal* and *unknown* behavior in time. This allows to analyze in specific, how these *normal*, *treacherous* and *unknown* neurons/clusters behave and evolve. The latter is a posterior separate study out of the scope of this work, relevant at the moment of predicting future behaviors, indicative of future *normal* behaviors or insider threat characteristics.

## 4.4 Discussion

The proposed algorithm describes an unsupervised clustering technique through the interrelation of a two-phase process: (1) learning, deletion and insertion procedures and (2) community labeling.

The adaptive learning rate allows the algorithm not only to consider data inputs that genuinely resemble to the existing ones for that BMU, but also to increase its specificity at every step, through decreasing the learning rate. Although this is an important enhancement towards

an online detection strategy, it may have some drawbacks. For the most part, it makes it significantly data-dependent. This issue may be sensitive when dealing with evolving data-streams, new monitoring points of observation, new attributes to consider, among others. For instance, if data evolve drastically into new normal behaviors and new anomalies, most of the too-specific existing neurons will not be good candidates, therefore increasing the number of insertions and topology of the network.

Nevertheless, the insertion neuron procedure benefits from being non-dependent of arbitrary *a-priori* parameters. By comparing every distance of the data input $x$ with its $bmu_1$, against both, a dynamic neighborhood threshold and severities contextual data; the algorithm guarantees those outliers (e.g., anomalies) are not to be absorbed by either well-adapted neurons or neurons at their early adaptive stage. This design is an advantage for the latter set of neurons, which in case of absorbing that input, would increase their local error.

Moreover, the *anomalous* or *treacherous* neurons will, by their nature, be part of sparse regions and will have very few *wins* in time and therefore, likely to be deleted. To treat this matter, the concept of *reservoir* has granted the possibility of saving that information for further treatment.

Frizke's approach (Algorithm 5) for deletion considers the age of the edges, namely if the age between two neurons is older than a predefined parameter, the edge is deleted. The subsequent step of the algorithm is to eliminate all isolated neurons after this edge deletion. Others [44], [50] have included a density parameter, considering also the number of times the neuron has "won" (i.e., BMU) an input. Additionally, Sun et al. [126] considered not only this information but also the average distance of every neuron $i$ towards their neighbors, $dist_i$, compared with a threshold.

Although the update of this value is considering the evolution of the network, it has some drawbacks when the anomalies have a very different distribution than the normal behavior. Their threshold is calculated by averaging all neurons scores $score_i = \exp^{-dist_i}$. Therefore regions where neurons have distant neighbors will decrease this averaged threshold. An example of this behavior obtained for 5 iteration cycles and an insert $\lambda$ value of 4, where this strategy caused the deletion of neurons with 273 or 163 winning times, due to the existence of scarce regions in the graph.

Experiments have shown that the evolution behavior of the *reservoir* is that it adapts slowly. As it is intended to keep neurons that would have been deleted otherwise, under the latter criterion they have low *winning* times $H_n$. These could be neurons that have absorbed anomalous data or new normal behaviors at an early stage. In both cases, if they are probably returning into the network, they will most likely be in the lower *tail* of the $H$ *win* times network distribution. Consequently, they are possible candidates for the deletion at the next iteration, as seen in the example from algorithm's output for *reservoir* management. This particularity, of course, can be overcome if those neurons absorb new inputs before the deletion process arrives. Finally, if they do not, they will return to the *reservoir*.

## 4.5 Conclusion

The proposed framework tackled the insider threat detection through an efficient and adaptive online learning and labeling schema. The procedure extends previous works by means of a better adaption for the insider threat monitored data. These adapting properties consisted of providing new insertion and deletion criteria, by using contextual information to increase the labeling precision and the quality of the clustering for anomaly detection of the insider threat.

Moreover, the proposed model adjusts itself according to the different scenarios in Section 6.2, fitting their data accordingly in the graph $\mathcal{G}$. From the experimental study in Chapter 6, the OSAD detection approach effectively reduces the FPR while maintaining good quality clustering values and not dismissing anomalous/treacherous data. This result is a definite advantage to consider because it means the model can be effectively used for different employees and different feature subsets while maintaining its clustering performance.

Additionally, the overall framework allows the extension of both employee's actions to be executed, along with the plausible augmentation of new attributes or features to monitor. This characteristic makes the proposed work generic for an anomaly detection procedure of the insider threat.

Furthermore, it incorporates a security detection procedure, by means of removing obsolete neurons, which are later classified according to their severity. This scheme is an improvement since it allows the network to evolve according to new input patterns. Additionally, the collected data in the *reservoir* can be interpreted as a kind of memory that may be useful again under specific conditions, for example when new inputs are too distant from all existing neurons. Therefore, these "obsolete" neurons preserve the knowledge of previous situations, and they could be taken for evaluation by an expert for future decisions, playing a role in a longer learning cycle of the employees or profiles in a company.

Chapter 5

# Monitoring and Implementation

**Contents**



## 5.1 Introduction

In the Chapter 4, the novel anomaly detection framework was described in detail, including the principal procedures required to apply its methodology. A further expansion of this research is the implementation of this methodology using a suitable tool. However, it is required to find such a tool in order to extend it with the proposed functionalities.

This chapter firstly presents the implementation details for the simulator described in Section 3, and the considerations taken for the generation of the employee's actions.

Secondly, it presents the implementation details of the proposed anomaly detection framework employing monitoring agents in a two-layer strategy. First, the MMT tool, a modular, extensible monitoring DPI [135] used to verify properties of network protocols. This tool is flexible enough to be used with other purposes than testing network protocols, making it suitable for implementing the proposed solution. In addition to this tool, this chapter presents the deployment and implementation of another monitoring agent, which is capable of retrieving

information based on the CSC's IaaS-based assets. These two first modules are part of the monitoring section, consequently received by the detection engine implementation.

Thirdly, we give details relative to the stream data treatment from the preprocessing stage of the anomaly detection.

The general implementation with all the components is shown in Figure 5.3.

## 5.2 Implemented cloud environment

The presented anomaly detection solution in Section 4 described in detail the data analysis for all the collected data. From an implementation point of view, this design follows a centralized architecture, where all CSC's nodes are monitored and all their correspondent data are collected to a behavior controller.

Our proposal follows then a cloud-based implementation in the open source cloud platform OpenStack [98]. OpenStack integrates many open source or proprietary software systems to perform different tasks. The management system, depicted in Figure 5.1, has a number of asynchronously connected modules, as well as a number of optional components to provide layer-3 routing, accounting, among others. Each module maintains its own database, and communicates through a persistent queue service [107].

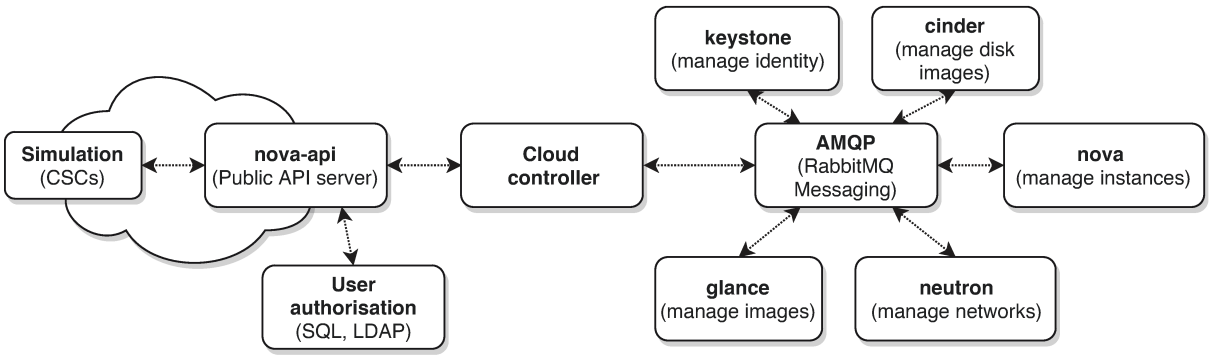



Figure 5.1 – Openstack services interaction with simulator

Moreover, every OpenStack service exposes its own Python API allowing to interact with the different services, scheduling actions for each one of them.

The experimental cloud platform used was a physical server running a virtual machine for running cloud management components. The OpenStack platform was installed on an Ubuntu 14.04 OS, on top of a VirtualBox [133] hypervisor. The following entities which the simulation interacted with, are described in Table 5.1.

Table 5.1 – Instance specifications

| Instance | OS | Size | RAM | Disk |
|---|---|---|---|---|
| VM0 | CirrOS | 12.65 MB | 512 MB | 1 GB |
| VM1 | CirrOS | 12.65 MB | 1024 MB | 1 GB |
| MMT | Ubuntu | 5.36 GB | 3.9 GB | 12 GB |

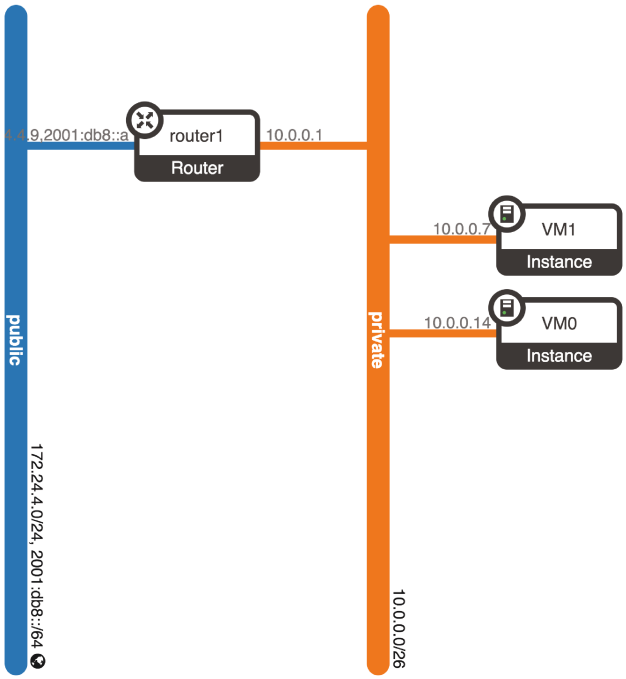



Figure 5.2 – Openstack network topology

## 5.3 Developed Framework

As seen in Figure 5.3, the framework is subdivided into two main sections: on one side it is the simulator component and on the other the detection component. In detail, the second concerns the retrieval of the attributes to analyze, pre-processing of data and the actual detection.

### 5.3.1 Simulated data

For every cloud service, the API was utilized to generate real actions towards the CSP.

#### 5.3.1.1 Actions

We used the libraries related to the OpenStack API, to simulate each of the `agent`'s (employees) calls to the CSC's `assets`. We also followed the documentation of the OpenStack server and images' possible status, across the simulated actions, resulting in the following actions (more information regarding the OpenStack states can be found in [99]).

- **List** retrieves a list of the VMs details.
- **Terminate** deletes a VM.
- **Reboot** reboots a VM.
- **Snapshot** generates an instantaneous copy of an instance. They contain an image of the state of the file-system at the moment that the snapshot is taken. A snapshot of an instance can be used as the basis of an instance and booted up at a later time.

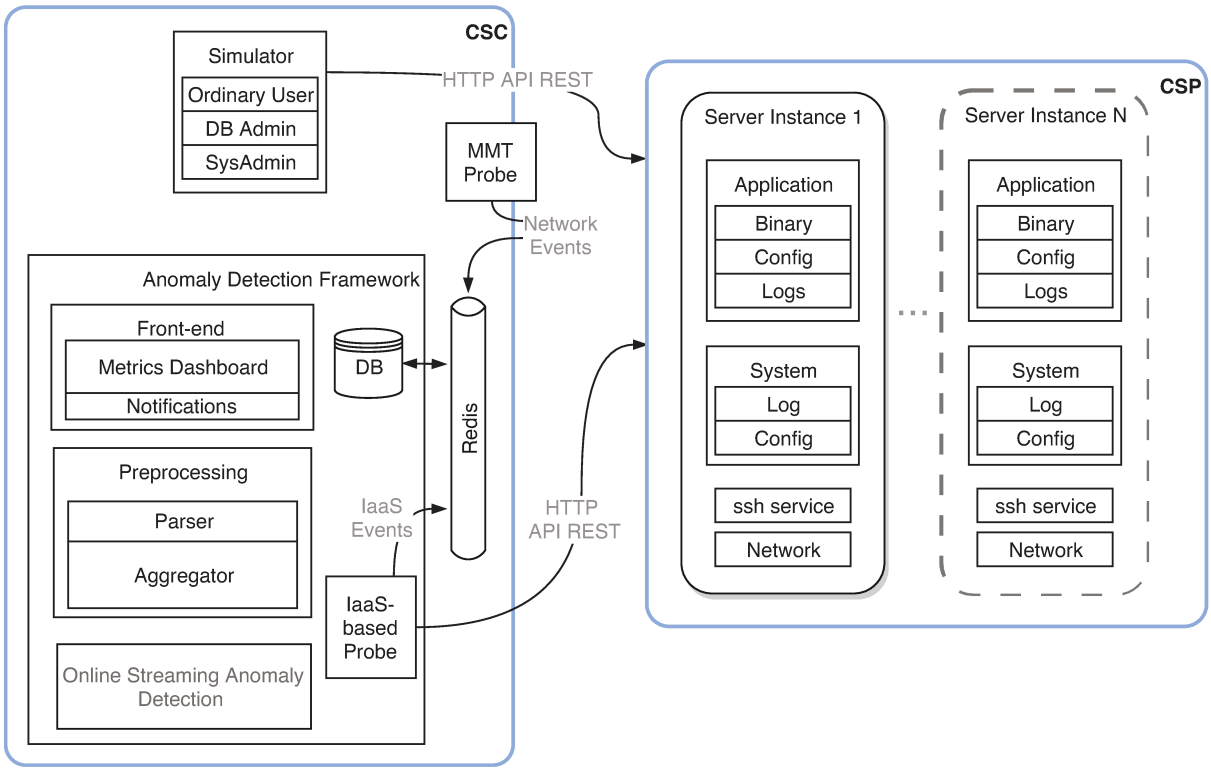



Figure 5.3 – Implementation of the anomaly-based detection framework

- **Stop** shuts down a VM.
- **Download** generates a bootable file-system that contains an operating system (Image).
- **Security rule modification** adds or eliminates a security rule (to open or close a port for a particular protocol, respectively).
- **Change server password** modifies the password of a VM.

#### 5.3.1.2 Implementation

The generic functions of the implemented actions are represented in Figure 5.4, where for each action to perform towards the OpenStack cloud, there are tasks, status checks and waiting functions.

**Action tasks** are related to the specific set of API calls to be done to the OpenStack cloud to complete the simulated action.

**Status checks** are performed before every task, to monitor the current status of the instance to be called (if possible do to so). This serves also to monitor the current status of the instance, which may be in a different current internal process than the allowed status to initiate the task. For example, if the action to perform is a *reboot*, the possible allowed state should be `ACTIVE`. This means that if the instance is currently `SHUTDOWN` or it does not exist, additional tasks should be triggered in order to start the main action.

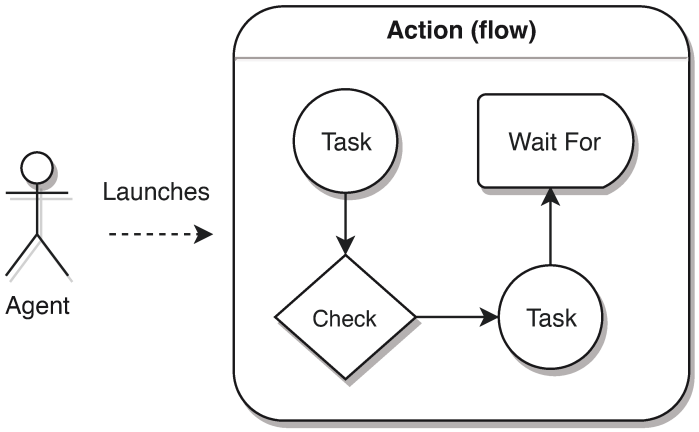



Figure 5.4 – Simulation action's main components towards OpenStack cloud

**Wait functions** are triggered after every task, to monitor the state of the instance called and to wait for its internal set of states to reach the target state of the task. For instance, if the task is to *stop* the instance, the target state to monitor and wait for, is `SHUTDOWN`. Afterwards, the simulation is performed to execute another action, following the proposed sequence of actions for that employee.

Two examples of this internal process are depicted in Figure 5.5. Note that all API calls towards the OpenStack cloud include simulation attributes embedded in the headers of the HTTP packets. This allows to differentiate all the task calls for a specific action task, with respect to to the main action task.

Later on, the HTTP flows are being collected by the monitoring probes described in the following Section 5.3.2.

### 5.3.2 MMT and IaaS monitoring probes

**Network probe** This is determined by the MMT monitoring kit which provides the core packet processing module to the framework. It consists in a C library that relies on DPI/DFI techniques in order to obtain information about the network or the application being analyzed. This module has the ability to inspect and analyze protocol fields' values, QoS-related network parameters, among others. In addition, this tool was designed to work both as an online monitoring tool or by analyzing structured information generated by the application or protocol under study (e.g., traces, logs). Finally, this module also includes an architecture that allows the expansion of the capabilities in order to analyze new protocols and applications, as well as an open API that allows the integration of third-party probes.

**IaaS-based probe** This serves as a contribution of the present thesis, as a monitoring agent for host-based data. It is a piece of software written in Python that uses an active monitoring approach to capture data concerning the cloud infra-structure of the tenant. It retrieves information related to the states of the VMs, the images, along with total memory use and Virtual CPU (vCPU) usage, among others.

These two probes periodically store the collected information, through a queuing communication channel.

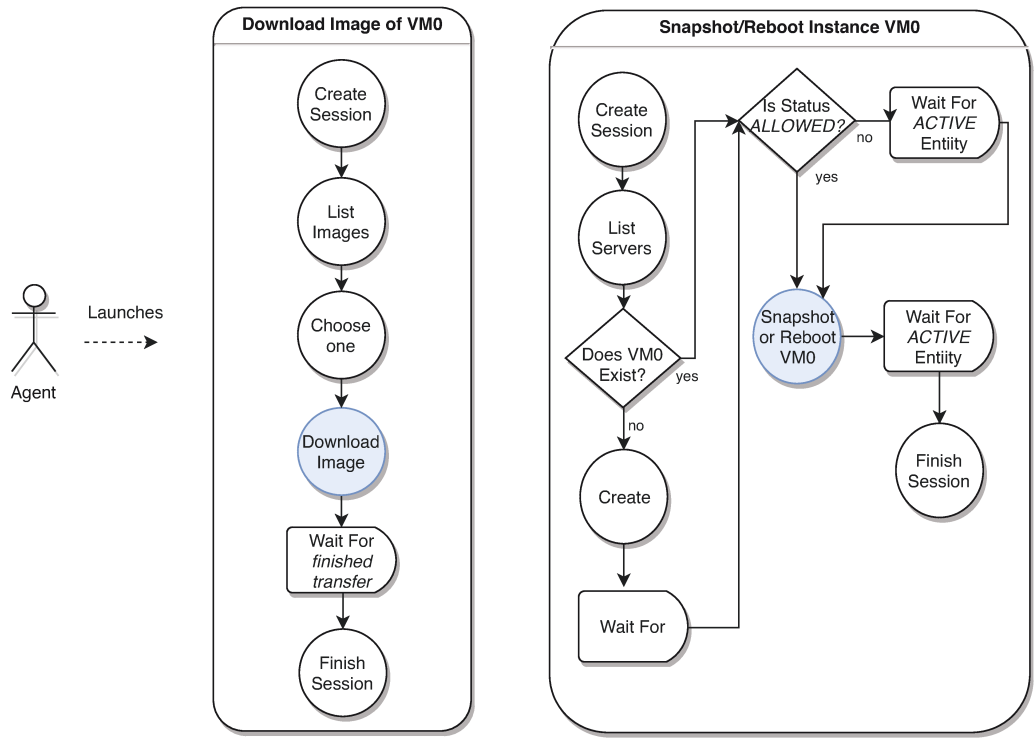



Figure 5.5 – Example of actions *Download*, *Reboot*, *Snapshot* for simulation action's flow towards OpenStack cloud

### 5.3.3 Communication and storage

This is the shared communication channel of all the components of the solution. This module is used to transmit messages and events from one component to another. It has been implemented as an internal message service running on an interface of the MMT machine in a local network (simulating a company) in the OpenStack cloud. The other modules of the solution connect to this service and can subscribe to published messages on the channel. This channel exchanges take place in a "monitoring" internal network, which is dedicated only for this matter, so it does not affect the traffic measurements of the tenant's activities.

Subsequently, these two event types are stored in a MongoDB [89] database for future retrieval of the detection module.

### 5.3.4 Preprocessing and feature analysis

In the Figure 5.6 the main workflow of the framework is presented, showing the main inputs and outputs. This section is comprised in two main steps, namely events' handling and events' aggregation.

#### 5.3.4.1 Events' handling and processing

The events are retrieved from the database and are treated by the `Events Handler`, which parses the incoming `Network Events` and `IaaS-based` events and constructs two objects: `Network Session` and `IaaS Session`, correspondingly.

**Network Events** Every `Network Event` or flow corresponds to a single action from the employee generated by the simulation, therefore we construct the session that corresponds to the list of actions generated by the simulation tool. This is done by means of keeping in memory the incoming flows and collecting the following respective attributes in Table 5.2.

| `Network Event` | `Network Session` | Calculation |
|---|---|---|
| Start Timestamp | Start Timestamp | event.start_timestamp[first] |
| | Session Time | event.start_timestamp[last] - event.start_timestamp[first] |
| | End Timestamp | event.start_timestamp[last] + event.data_transfer_times[last] |
| Total Bytes | Total Bytes | sum(event.total_bytes) |
| Total Packets | Total Packets | sum(event.total_packets) |
| Response time | Response time | sum(event.durations) |
| UL Data Volume | UL Data Volume | sum(event.ul_data_vols) |
| DL Data Volume | DL Data Volume | sum(event.dl_data_vols) |
| Source bytes | Source bytes | sum(event.source_bytes) |
| HTTP Method | POST Method Count | sum(event.post_method_counts) |
| | GET Method Count | sum(event.get_method_counts) |
| | PUT Method Count | sum(event.put_method_counts) |
| | DELETE Method Count | sum(event.delete_method_counts) |
| Transaction count | Transaction count | sum(event.transaction_counts) |
| Data Transfer time | Duration | sum(event.data_transfer_times) |

Table 5.2 – Network features

Additionally, the user-agent attribute from the HTTP contains the following simulation attributes.

| Simulation Attributes | | |
|---|---|---|
| Asset | Number of actions | Simulation location |
| Profile name (employee's name) | Event Id | Label |
| *ith*-Action | Simulation timestamp | |

The construction of the object `Network Session`, is based on the aggregation of several `Network Event` objects. This is done by the designation of the tuple (profile name, event id, asset) and the aggregation of the network attributes until any of the following conditions is satisfied:

- *Complete session*: is considered when the *ith*-action attribute is equal to the *number of total actions* within the session.
- *Timeout*: is considered when an event is not completed (i.e., previous condition is not met), i.e., when TCP/IP packets are re-transmitted, fragmented, or when the analysis of the data stream of actions is delayed. We utilized a timeout of four minutes in our implementation.

Moreover, from the *Simulation timestamp* and *Simulation location* we include two attribute

values, namely 0 or 1, describing whether the simulated actions were intended to be within the working hours and within the organization's dependencies, respectively.

Lastly, the *Label* is kept for the purpose of evaluating the proposed detection solution.

**IaaS events** Every `IaaS event` or flow corresponds to a single active monitoring check made to retrieve information from the OpenStack cloud. This type of event is not related to any specific employee and thus it is used for future merging with the `Network Session` objects. The final attributes are formerly retrieved from two main categories:

- *Instance state*: describes the current stable (not transitional) state of the VM. That is, if there is no ongoing compute API call (running task), the instance state should reflect the VM state expected by the CSC. When combined with task states, a better picture can be formed regarding the server's "health" and progress;
- *Task state*: represents what is happening to the instance at the current moment and allow a better view into what a server is doing. These tasks can be generic, such as "spawning", or specific, such as "powering-off".

The latter becomes relevant at the moment of performing an active monitoring with a non-invasive manner. We make use of a combination of these two categories of attributes, in order to benefit from two sources instead of one that may miss an occurrence.

By this compound methodology, it was possible to retrieve absent information from the *VM state* in the percentages depicted in Table 5.3 following a 10 second active check configuration. This was obtained by combining data from both data subsets, where some *VM state* attributes resulted in a constant value of 0, meaning they were not foreseen by our monitoring configuration. A example of this case is `RESTART` state being updated in the `Server Reboot Mean` by the finding of *Task state* powering-on and powering-off.

Table 5.3 – Percentage of traceable records with merging methodology

| Profile | Instance action | Task state | Recovered records | Percentage |
|---|---|---|---|---|
| Employee 1 | Server Reboot | reboot | 1037 | 42.378 |
| | Image Snapshot | createImage | 189 | 100.000 |
| | Server Terminate | delete | 39 | 100.000 |
| | Stop server | stop | 162 | 100.000 |
| | Server Change Password | changePassword | 172 | 100.000 |
| Employee 2 | Server Reboot | reboot | 2550 | 41.050 |
| | Image Snapshot | createImage | 195 | 100.000 |
| | Server Terminate | delete | 40 | 100.000 |
| | Stop server | stop | 167 | 100.000 |
| | Server Change Password | changePassword | 180 | 100.000 |

The final list of attributes or features can be found in the first column of Table 5.4.

#### 5.3.4.2 Events Aggregation

The second step is the aggregation of the data by means of merging of the `Network Session` and `IaaS Session` datasets. This is done following the methodology in Algorithm 13 and calculated

as shown in Tables 5.2 and Table 5.4. Most features are calculated following the mean or average, where "$\circ$" is the Hadamard product (element-wise) of each of the events attribute lists. This is because it is intended to capture the tendencies among all aggregated sessions.

Two other mathematical operations are performed to the rest of the features, followed by the variance of the session individual features treated as Boolean. The first case relies on the intention to capture the variability rather than the general tendency, because abrupt changes could signify anomalies. The second case (e.g., security rules' modification) aims to capture any alteration made to the network port rules. This means that for the scope of this work the feature is not intended to gain information on how much the rules were modified, rather than the fact that this modification took place. The intuition behind the latter is that it could only take one single modification (e.g., an addition of a rule) to perform an attack.

**Algorithm 13** Network-IaaS Events' Merging

**Input:** Network Session List, IaaS Session List per profile
**Output:** Network-IaaS Sessions per profile

1: **for all** $net\_session \in NetworkSessions$ **do**
2: $init\_ts \leftarrow net\_session.init\_timestamp$
3: $end\_ts \leftarrow net\_session.end\_timestamp$
4: $iaas\_sessions \leftarrow$ FilterIaaSRecords$(init_ts, end_ts, iaas_sessions)$
5: **if** iaas_sessions not empty **then**
6: $iaas_{aggregated} \leftarrow$ AggregateIaaSSession$(net_session, iaas_sessions)$
7: $netiaas\_sessions \leftarrow$ AddNetworkIaaSSession$(net_session, iaas_{aggregated})$
8: **else**
9: Continue
10: **end if**
11: **end for**

Table 5.4 presents all the features captured from the IaaS-based probe and it is divided according to the monitored instance, namely, Image, Sever and Network. Additionally, an overall metric concerning the memory usage of all the instances is introduced.

As mentioned in Section 4.3.1, some of the features share no information due to several factors. First, the sampling mechanism may not have the adequate resolution to retrieve data, due to missing the assets' states applying the active monitoring. Second, the absence of an action that reveals information towards this feature. Nevertheless, the monitoring agent is implemented and capable of retrieving future actions that may change the empty-value attributes.

**Additional representations** In order to distinguish anomalies more efficiently, three attributes have been proposed, based on the previous knowledge concerning the security threats intended to detect. They have been introduced due to two main reasons.

First, the profitability of information knowledge within the enterprise. In other words, relevant information can be retrieved from the employees in charge of the common practices and security policies of the company (e.g., human resources, security and/or operation administrative) in what respects to job roles, and more importantly, regarding the company's cloud assets (e.g., cloud administrator, system administrator). For the former group, utilizing this already-existing information is convenient. The second group has the knowledge information on the configuration of the cloud implementation. This configuration could imply an IaaS ser-

| `IaaS Event` | Calculation for `IaaS-Session Event` object |
|---|---|
| Image Preparing Mean | mean(image_preparing_counts ∘ 1/image_total_counts) |
| Image Failed Mean | mean(image_failed_counts ∘ 1/image_total_counts) |
| Image Queued Mean | mean(image_queued_counts ∘ 1/image_total_counts) |
| Image Pending Delete Mean | mean(Image pending_delete_counts ∘ 1/image_total_counts) |
| Image Other Mean | mean(image_pother_counts ∘ 1/image_total_counts) |
| Image Killed Mean | mean(image_killed_counts ∘ 1/image_total_counts) |
| Image Unknown Mean | mean(image_unknown_counts ∘ 1/image_total_counts) |
| Image Delete Mean | mean(image_delete_counts ∘ 1/image_total_counts) |
| Image Snapshot Mean | mean(image_snapshot_counts ∘ 1/image_total_counts) |
| Image Active Mean | mean(image_active_counts ∘ 1/image_total_counts) |
| Image Total Mean | mean(image_total_counts ∘ 1/image_total_counts) |
| Total Image Variance | var(image_total_counts) |
| Server Unknown Mean | mean(server_unknown_count ∘ 1/server_total_counts) |
| Server Hard Reboot Mean | mean(server_hardreboot_count ∘ 1/server_total_counts) |
| Server Paused Mean | mean(server_paused_count ∘ 1/server_total_counts) |
| Server Stopped Mean | mean(server_stopped_count ∘ 1/server_total_counts) |
| Server Verify Resize Mean | mean(server_verifyresize_count ∘ 1/server_total_counts) |
| Server Resize Mean | mean(server_resize_count ∘ 1/server_total_counts) |
| Server Suspended Mean | mean(server_suspended_count ∘ 1/server_total_counts) |
| Server Delete Mean | mean(server_delete_count ∘ 1/server_total_counts) |
| Server Rebuild Mean | mean(server_rebuild_count ∘ 1/server_total_counts) |
| Server Rescue Mean | mean(server_rescue_count ∘ 1/server_total_counts) |
| Server Other Mean | mean(server_other_count ∘ 1/server_total_counts) |
| Server Prep Resize Mean | mean(server_prepresize_count ∘ 1/server_total_counts) |
| Server Error Mean | mean(server_error_count ∘ 1/server_total_counts) |
| Server Queue Resize Mean | mean(server_queueresize_count ∘ 1/server_total_counts) |
| Server Password Mean | mean(server_password_count ∘ 1/server_total_counts) |
| Server Build Mean | mean(server_build_count ∘ 1/server_total_counts) |
| Server Shutoff Mean | mean(server_shutoff_count ∘ 1/server_total_counts) |
| Server Reboot Mean | mean(server_reboot_count ∘ 1/server_total_counts) |
| Server Total Mean | mean(server_total_count ∘ 1/server_total_counts) |
| Server Active Mean | mean(server_active_count ∘ 1/server_total_counts) |
| Total Server Variance | var(server_total_counts) |
| Security Rules Modification | 1 if var(security_rules) $>$ 0 else 0 |
| Total Memory Mb Usage | std(total_memory_mb_usages) |

Table 5.4 – IaaS-features

vice model, where this group of people would know the internal network topology and rules' composition. If the company's services also include applications on top of the IaaS, practical information can be retrieved on what respects to acceptable behaviors, such as CPU loads, applications' up-time, number of VMs or replicas that should be active, port access rules, among others.

Second, the *know-how* already seen in many monitoring and information logging references

in the literature, such as the RFC 3164[1]. The latter presents a set of recommendations for categorizing messages with a severity indication for that message. More specifically, they present predefined scales of integers to define different severity levels (e.g., 0 to 9 where 0 is informative, and 9 is critical). These scales serve as information for the specific group of people that derived them, and therefore depend on the relevance they give to each of the components.

These monitoring and auditing considerations have served as inspiration for the below-proposed schema. Moreover, in the proposed case scenario of a company with their assets in the cloud, the three aggregated features that are proposed, consider the help of the above mentioned decisive players. Consequently, the below-detailed severity definitions and scale factors are adjustable to each company's needs and experts opinions.

That being said, these three features cannot be instantaneously checked when collecting the data since it would demand a higher level of monitoring complexity.

In detail, not all severity levels have the same degree of urgency. Some require immediate human attention, some require eventual human attention, and some point to areas where attention may be needed in the future.

Consequently, we propose three severity categories, with respect to the instance to which they are related. They are defined as:

- *Server Severity*: describes the presence of a compromise for a VM in either availability, confidentiality or integrity.
- *Image Severity*: describes the presence of a compromise for an Image in either availability, confidentiality or integrity.
- *Network Severity*: describes the presence of a compromise for a property of the network in either availability, confidentiality or integrity.

Hence, they are created at the moment of merging the collection of `IaaS Events` with a specific `Network Event`. They are calculated as the weighted sum of the `IaaS Event` features, where the weigh is a factor dependent on the attribute, as seen in Table 5.5 and Table 5.6.

## 5.4 Discussion

With respect to the presented IaaS-based probe, it is worth mentioning the absence of monitoring tools from a tenant point of view. This is the case of existing tools such as Amazon's Cloudtrail[2], where CSC can monitor the anomalies with respect to the assets in the cloud for Amazon EC2 instance changes, EC2 large instance changes, console sign-in failures, authorization failures, Identity and Access Management (IAM) policy changes, to name a few. Then, there is a benefit – though not free – to monitor anomalies in companies' assets in the cloud. In contrast, an OpenStack implementation does not have this monitoring feature by default, and it should be agreed for CSC's and CSP's parts.

Our proposed schema, on the other hand, does not have this possibility because OpenStack services are entirely independent; there is no service that monitors the access to their endpoints. Many API servers log the requests when debug logging is enabled, but even so, all these logs occur in a CSP's domain.

[1]The BSD Syslog Protocol

[2]Service that enables compliance and security auditing by logging all actions taken by users towards the CSC's assets

Table 5.5 – Severity factors for Image and Server

(a) Server severity

| Valid Server IaaS attributes | Factor |
|---|---|
| Server Active Count | 0 |
| Server Build Count | 1 |
| Server Rebuild Count | 1 |
| Server Queue Resize Count | 1 |
| Server Prep Resize Count | 1 |
| Server Resize Count | 1 |
| Server Verify Resize Count | 1 |
| Server Reboot Count | 2 |
| Server Hard Reboot Count | 2 |
| Server Password Count | 3 |
| Server Paused Count | 3 |
| Server Suspended Count | 4 |
| Server Migrating Count | 4 |
| Server Rescue Count | 5 |
| Server Delete Count | 5 |
| Server Unknown Count | 5 |
| Server Stopped Count | 5 |
| Server Shutoff Count | 7 |
| If sum of Servers Count is 0 | 9 |
| If sum of Active Servers' Count is 0 | 9 |

(b) Image severity

| Image IaaS attributes | Factor |
|---|---|
| Image Active Count | 0 |
| Image Queued Count | 2 |
| Image Preparing Count | 2 |
| Image Deactivated Count | 4 |
| Image Unknown Count | 5 |
| Image Failed Count | 5 |
| Image Delete Count | 9 |
| Image Pending Delete Count | 9 |
| Image Killed Count | 9 |
| If Sum of Image Active Counts is 0 | 9 |
| If Total Image Total Counts is 0 | 9 |

Table 5.6 – Severity factors for network in the cloud

| Network attributes | Factor |
|---|---|
| Security rules' Count | 9 |

This situation is an issue regarding the relevant plausible data to capture because much information is lost during an active monitoring mechanism (e.g., a log-in failure action may happen in less than a second, while active monitoring probes are retrieving information every 5 seconds).

Instead, a more effective mechanism would be to collect the log data from cloud services and passively foresee every single change in the CSC's activities. In this sense, our contribution aims at allowing CSC's to monitor their assets in OpenStack with similar efficiency, while having limited information.

Furthermore, the overall monitoring and detection framework has been designed with a single tenant use in mind. This solution is intended to give every company whose desire is to detect the insider threat, the possibility to monitor their assets not only in a single cloud application but also multi-cloud (i.e., one tenant with multiple cloud services connected). In its current form, this makes sense for organizations with a small number of single or multi-cloud services, to consolidate their security management and keep track of anomalies in time.

From a technical perspective, what needs to be considered is the scalability of the framework once a higher number of security metrics are observed or aggregated. These could be in the aim of adding not only IaaS-based events but also, monitoring the availability, confidentiality or integrity of applications embedded in the IaaS assets. For instance, the wish to monitor a new module (e.g., an application on top of IaaS) may require the addition of another probe and a new set of arrangements for the calculation of simple or aggregated attributes.

For the solution proposed, we have chosen to deploy the network and IaaS-based probes on the organization's side for three main reasons.

First, even if the collected traffic at the organization's gate is considering all network communications that could be non-cloud related, the MMT tool allows the filtering of this traffic and permits the reporting of only the protocols and destination of interest. Like so, we have filtered only the HTTP network flows, along with the destination IP of the machine hosting the cloud. Nevertheless, the architecture benefits from the adaptability of the MMT tool, in the direction of detecting anomalies related to network protocols (e.g., such as CERT's dataset, which presents insider threat scenarios using log-ins from different ports and protocols).

Second, this benefits within the scope of security. By having a local implementation, the enterprise has additional protection for the monitored data at storage, whose communication does not go through the public internet and the access to it is under the security policies of the company. This policy contrasts the idea of having the monitoring probes, storage and detection engine as additional assets in the cloud, which makes them more visible to possible cloud administrators that could perform malicious actions through the direct access to them.

Lastly, we also consider the billing aspect by means of externalizing all the monitoring traffic from the different probes to the organization's premises. This benefits from having sensors that are actively using resources, such as I/O read/write, bandwidth usage and storage capacity.

## 5.5 Conclusion

In this chapter, the details of the implementation were explained, both for the simulator in Chapter 3, as well as for the anomaly detection framework in Chapter 4. This work was done by proposing a set of simulated activities or actions, that considered the internal states of the assets involved, in order to have an automatic derivation of generated actions. Also, we extended the existing monitoring sensors by means of deriving a new IaaS-based probe. Finally, we presented the architecture of the collected events, along with the preprocessing management to serve as a precise input for the detection engine.

In the next Chapter 6, we present the experimental results obtained with the use of the developed framework.

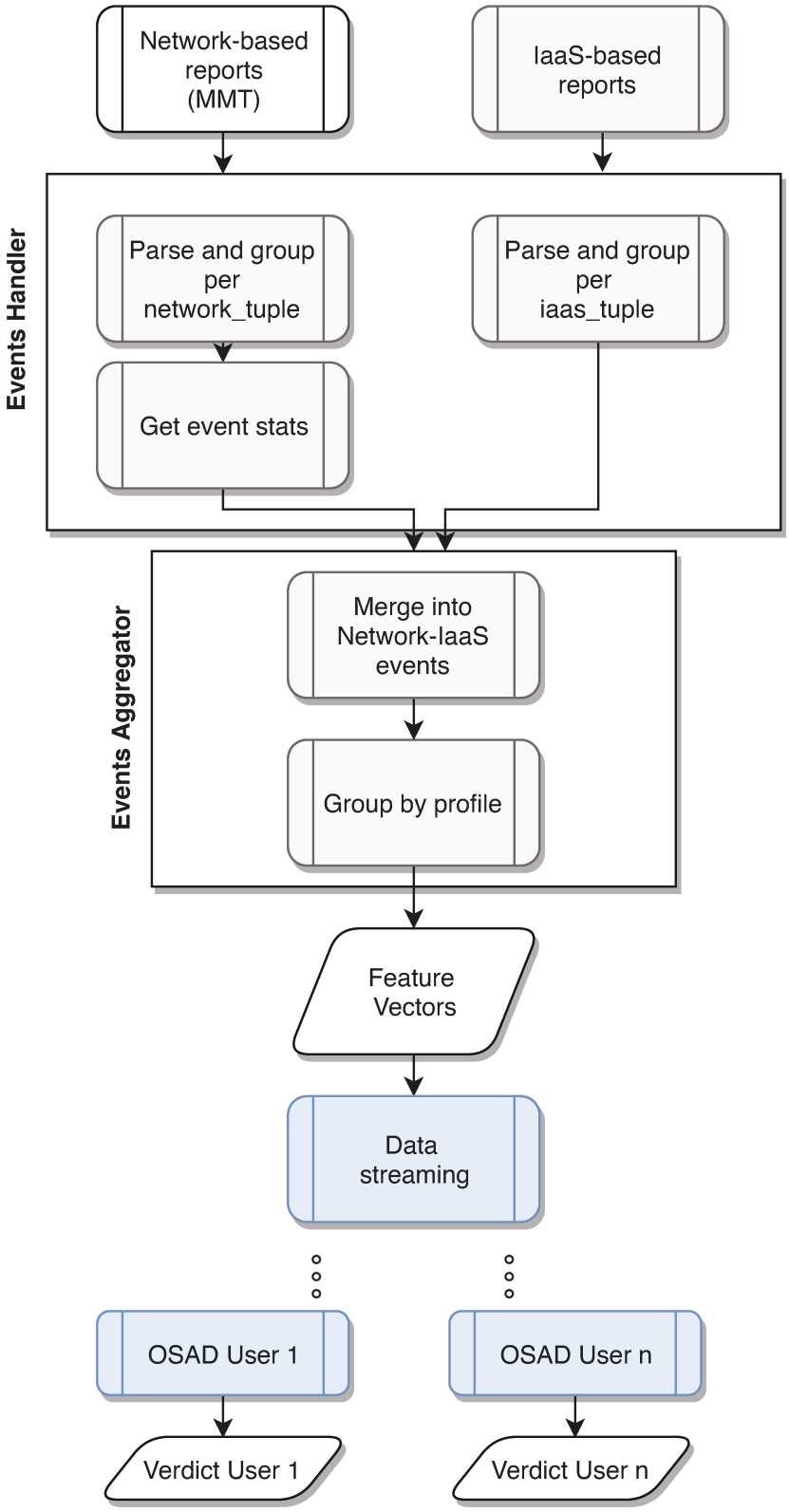



Figure 5.6 – Workflow for the detection framework

Chapter 6

# Experimental evaluation

**Contents**



## 6.1 Introduction

This section presents the experiments which have been carried out to measure the quality of the proposed algorithm of Section 4. For this reason, the section contains the results of a series of experiments which have been carried to evaluate the effectiveness and usability of the developed framework, aiming to distinguish insider threat anomalies by considering different data attributes.

This is done by benefiting from the flexibility of the user model simulation presented in Section 3. A study of four detection scenarios is presented, each one of them relying on typical insider threat characteristics found in the literature and in the industrial reports.

Additionally, the section provides experimental results by means of comparing the proposed algorithm in Chapter 4 with different hyper-parameters, different distance metrics used in the GNG-based algorithm; different dataset imbalance ratios, distinct users, among others. These comparisons have been discussed through the evaluation metrics presented Section 4.2.5.

## 6.2 Detection scenarios

For the synthetic experiments, the following datasets have been created. The first two differ in two main characteristics: the user's lexicographical sequences of actions, and the use of contextual data to identify anomalies.

The four presented scenarios contain the same type of sequence anomalies, but rely on different data subsets for their detection.

### 6.2.1 Scenario 1: Using Network, IaaS features

The subset used in this scenario consists in all network-based and IaaS-based features described in Chapter 4 through the implementation described in Chapter 5.

### 6.2.2 Scenario 2: Using Network, IaaS and action sequences' features

The subset used in this scenario consists in all network-based and IaaS-based features described in Chapter 4 through the implementation described in Chapter 5. In addition to that set of features, this scenario includes the sequence of actions executed by the employee, provided by the insider threat model from Chapter 3.

### 6.2.3 Scenario 3: Using Network, IaaS and contextual features

Following the same characterization of the CERT dataset, where some anomalous actions occur out of the scope of *user-days* [142], [143], the elaboration of this dataset scenario extends the previous features (network-based, IaaS-based and sequences) to anomalies using this particularity, along with anomalous location.

### 6.2.4 Scenario 4: Using Network, IaaS, actions sequences' and contextual features

This scenario comprehends the features' set of action sequences, network-based and IaaS-based, along with the contextual working hours and location features. These two Boolean features are now randomly assigned to 1 whenever the label is of the data input, i.e., "seq" (anomaly). The purpose of these additional attributes is to see if these two extra features help improving the detection.

## 6.3 Experimental results

### 6.3.1 Analysis of the global error and learning procedure

The process of neuron's learning has been previously described in Section 4.2.3. The criterion of Fritzke's original GNG [43] was presented, along with the adaptive proposal of [44] described in Equations 4.1 and 4.2, respectively. As explained before, the latter consists in a variable that

updates itself for every time-step new input vector arrival and its winning neuron $bmu_1$, related to the $bmu_1$'s number of *wins*.

The hyper-parameters' configuration for all the algorithms was fixed to be able to compare how the scenarios affect the overall global error, along with the internal metric results. The values were used with a fixed step interval between neurons insertion $\lambda$ of 50, and an edge deletion of $age_{max}$ of 80. The learning rate trials were made with the values from [50], namely $\alpha_1 = 0.1$ $\alpha_2 = 0.01$, the original GNG [43] $\alpha_1 = 0.2$ $\alpha_2 = 0.006$, another arbitrary $\alpha_1 = 0.5$ $\alpha_2 = 0.1$. Additionally, the adaptive learning rate was evaluated, following Equation 4.2. For this example, each iteration has 1000 input samples, all results are averaged over 10 algorithm runs. Moreover, a reader can also see that the initial accumulated errors are different since the first two GNG neuron's prototypes are initiated randomly following a uniform distribution.

| ID | Profile | Feats | Mod. | Sil. Coef. | Neurons | Global E. | Local Acc. E. |
|---|---|---|---|---|---|---|---|
| 1 | Cloud Admin High Adaptive | 36 | 43 | 0.652 | 0.118 | 16432.2 | 0.799 ± 1.768 |
| 1 | Cloud Admin High Fix [50] | 36 | 43 | 0.637 | 0.235 | 26452.9 | 3.155 ± 9.909 |
| 1 | Cloud Admin High Fix [43] | 36 | 43 | 0.577 | 0.115 | 9342.74 | 0.904 ± 4.773 |
| 1 | Cloud Admin High Fix [0.5, 0.1] | 36 | 43 | 0.335 | -0.092 | 11654.1 | 0.745 ± 5.116 |
| 2 | Cloud Admin High Adaptive | 44 | 43 | 0.619 | 0.27 | 27770.4 | 1.174 ± 2.506 |
| 2 | Cloud Admin High Fix [50] | 44 | 43 | 0.669 | 0.198 | **32853.8** | 2.128 ± 4.724 |
| 2 | Cloud Admin High Fix [43] | 44 | 43 | 0.67 | 0.261 | 11992.2 | 0.98 ± 3.769 |
| 2 | Cloud Admin High Fix [0.5, 0.1] | 44 | 43 | 0.457 | -0.019 | 10165.7 | 0.894 ± 5.186 |
| 3 | Cloud Admin High Adaptive | 38 | 43 | 0.629 | 0.252 | 17668.7 | 0.869 ± 1.869 |
| 3 | Cloud Admin High Fix [50] | 38 | 43 | 0.626 | 0.19 | 28934.4 | 1.605 ± 3.155 |
| 3 | Cloud Admin High Fix [43] | 38 | 43 | 0.557 | **0.278** | 12064 | 0.659 ± 1.791 |
| 3 | Cloud Admin High Fix [0.5, 0.1] | 38 | 43 | 0.429 | 0.022 | 11423.1 | 0.689 ± 1.705 |
| 4 | Cloud Admin High Adaptive | 42 | 43 | 0.676 | 0.25 | 22096.5 | 0.919 ± 1.826 |
| 4 | Cloud Admin High Fix [50] | 42 | 43 | **0.686** | 0.26 | 31624.5 | 1.619 ± 3.147 |
| 4 | Cloud Admin High Fix [43] | 42 | 43 | 0.577 | -0.075 | 13587.6 | 0.765 ± 1.8 |
| 4 | Cloud Admin High Fix [0.5, 0.1] | 42 | 43 | 0.381 | -0.042 | 13627.8 | 0.737 ± 1.75 |
| 1 | Cloud Admin Medium Adaptive | 36 | 72 | 0.672 | 0.117 | 60800.8 | 1.177 ± 7.076 |
| 1 | Cloud Admin Medium Fix [50] | 36 | 72 | 0.733 | 0.048 | 77880.4 | 3.386 ± 14.239 |
| 1 | Cloud Admin Medium Fix [43] | 36 | 72 | 0.642 | -0.003 | 33747.8 | 1.276 ± 9.813 |
| 1 | Cloud Admin Medium Fix [0.5, 0.1] | 36 | 72 | 0.409 | 0.061 | 44461.7 | 1.091 ± 9.212 |
| 2 | Cloud Admin Medium Adaptive | 44 | 72 | 0.698 | 0.152 | 74072.5 | 1.554 ± 8.217 |
| 2 | Cloud Admin Medium Fix [50] | 44 | 72 | **0.743** | 0.02 | **99998.7** | 6.608 ± 30.24 |
| 2 | Cloud Admin Medium Fix [43] | 44 | 72 | 0.603 | 0.199 | 38992.1 | 1.329 ± 9.028 |
| 2 | Cloud Admin Medium Fix [0.5, 0.1] | 44 | 72 | 0.499 | -0.049 | 46110.2 | 1.291 ± 9.679 |
| 3 | Cloud Admin Medium Adaptive | 40 | 72 | 0.685 | 0.005 | 67038.1 | 1.356 ± 7.567 |
| 3 | Cloud Admin Medium Fix [50] | 40 | 72 | 0.698 | 0.08 | 83235.4 | 5.2 ± 23.619 |
| 3 | Cloud Admin Medium Fix [43] | 40 | 72 | 0.58 | 0.029 | 42179.8 | 1.332 ± 9.638 |
| 3 | Cloud Admin Medium Fix [0.5, 0.1] | 40 | 72 | 0.408 | -0.023 | 45435.3 | 1.102 ± 9.303 |
| 4 | Cloud Admin Medium Adaptive | 44 | 72 | 0.714 | **0.283** | 75631.7 | 1.459 ± 8.555 |
| 4 | Cloud Admin Medium Fix [50] | 44 | 72 | 0.707 | 0.015 | 91396.5 | 5.065 ± 22.299 |
| 4 | Cloud Admin Medium Fix [43] | 44 | 72 | 0.648 | 0.191 | 48177.3 | 1.335 ± 9.482 |
| 4 | Cloud Admin Medium Fix [0.5, 0.1] | 44 | 72 | 0.491 | -0.028 | 38805.6 | 1.163 ± 9.275 |

Table 6.1 – Experimental results for different learning rates

The results of the learning and overall detection procedure for all different subset scenarios, can be seen in Figure 6.1, through the global error of the network for every iteration step. They depict the accumulated error, correspondent to the sum of all neurons' errors at that particular time step. The iteration number 0 corresponds to how the algorithm learned by passing through the data only once (e.g., online manner). Additionally, Table 6.1 depicts how the different learning rates affect the overall clustering quality and global error.

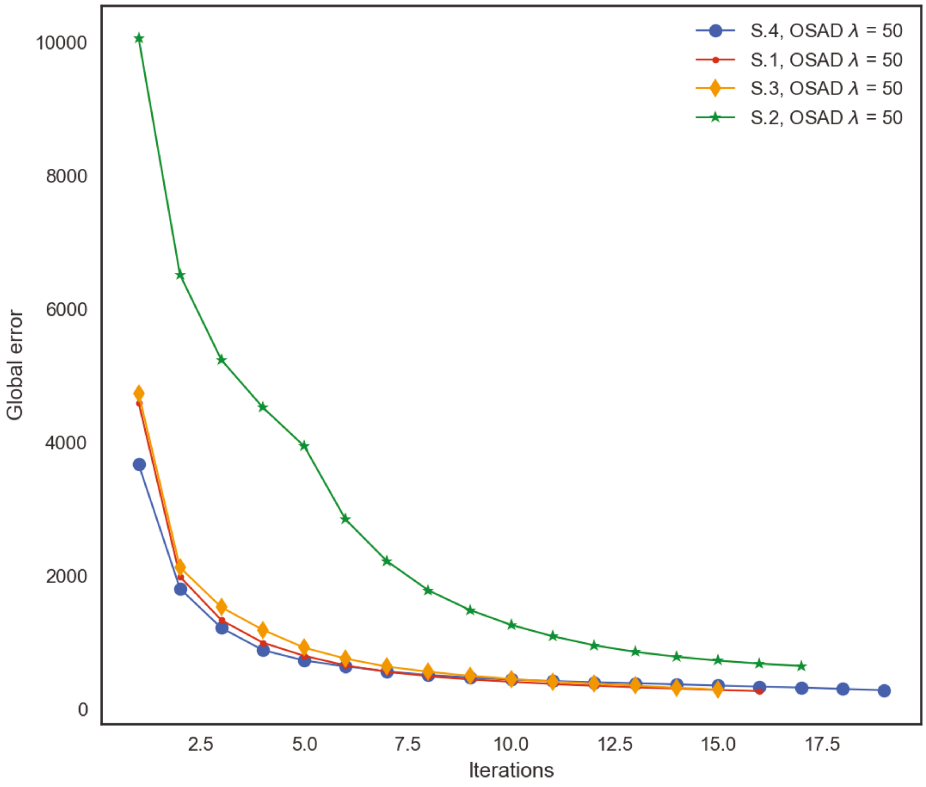



Figure 6.1 – Global accumulated error for all scenarios with the OSAD algorithm

Those findings display that the learning evolution for all scenarios increases in time, reaching to similar values at the end of the last iteration 18. This is not necessarily related to "what" the algorithm learned or if it properly clustered the different behaviors, however it means that the neurons have already acquired the input vectors' information, therefore the absorption of further similar behaviors is already contained in the graph.

As stated in the algorithm, this learning performance also relies on the previously mentioned insertion and deletion components, which are discussed in the following section.

### 6.3.2 Analysis of the insertion procedure

The process of inserting neurons whenever a criterion is met, has been previously described in Section 4.3.3.1. The criterion for the Fritzke's original GNG [43] was presented. As explained before, the latter consists in an adaptive threshold-based criterion for every time-step new input vector arrival and its winning neuron $bmu_1$, related to the $bmu_1$'s number of *wins* and its prototype $w_{bmu_1}$.

The presented set of experiments detailed in Table 6.2 are relevant to this matter. A set of 30 trials has been launched for each experiment (hence the values in mean ± standard deviation). The insertion rate was fixed to 50 and the number of offline training iterations was 10.

| ID | Profile | Time | Feats | Mod. | Sil. Coef. | Neurons | Global E. | Local Acc. E. |
|---|---|---|---|---|---|---|---|---|
| 1 | Cloud Admin High Fix | 13.987 | 36 | 0.435 | 0.187 | 18 | 15941.9 | 0.716±1.618 |
| 1 | Cloud Admin High Adaptive | 66.820 | 36 | 0.841 | 0.035 | **324** | 2276.62 | 0.012±0.042 |
| 1 | Cloud Admin Medium Fix | 18.031 | 36 | 0.51 | 0.159 | 27 | 41599.3 | 1.09±7.241 |
| 1 | Cloud Admin Medium Adaptive | 66.106 | 36 | 0.795 | 0.371 | 231 | 10678.5 | 0.01±0.036 |
| 2 | Cloud Admin High Fix | 12.926 | 44 | 0.392 | 0.07 | 18 | 24132.7 | 0.999±2.207 |
| 2 | Cloud Admin High Adaptive | 58.287 | 44 | 0.838 | 0.374 | 276 | 2562.72 | 0.016±0.051 |
| 2 | Cloud Admin Medium Fix | 16.509 | 44 | 0.439 | 0.337 | 27 | 39912.3 | 1.102±7.274 |
| 2 | Cloud Admin Medium Adaptive | 72.009 | 44 | 0.834 | 0.369 | 306 | 12816.3 | 0.01±0.033 |
| 3 | Cloud Admin High Fix | 12.158 | 38 | 0.386 | 0.251 | 18 | 20338.3 | 0.878±3.434 |
| 3 | Cloud Admin High Adaptive | 51.932 | 38 | **0.848** | **0.382** | 298 | 3200.21 | 0.008±0.019 |
| 3 | Cloud Admin Medium Fix | 20.524 | 40 | 0.473 | 0.205 | 27 | 35685.6 | 0.901±5.217 |
| 3 | Cloud Admin Medium Adaptive | 73.782 | 40 | 0.82 | 0.27 | 270 | 7128.76 | 0.01±0.043 |
| 4 | Cloud Admin High Fix | 17.482 | 42 | 0.43 | 0.115 | 18 | 22965.9 | 0.848±2.049 |
| 4 | Cloud Admin High Adaptive | 62.872 | 42 | 0.832 | 0.116 | 310 | 2601.34 | 0.014±0.029 |
| 4 | Cloud Admin Medium Fix | 16.528 | 44 | 0.508 | 0.222 | 27 | **43936.91** | 1.16±6.634 |
| 4 | Cloud Admin Medium Adaptive | 67.223 | 44 | 0.818 | 0.22 | 248 | 12663.3 | 0.014±0.058 |

Table 6.2 – Adaptive neuron insert comparison (proposed algorithm and original)

### 6.3.3 Analysis of the deletion procedure

The process of deleting neurons whenever a criterion is met has been previously described in Section 4.3.3.1. This procedure has a different behavior when utilizing different feature subsets. Such variations can be seen in Figure 6.2, where specifically in Figure 6.2b the deleted number of neurons per iteration is very low. In detail, scenarios 2 and 4 show higher values than the others. This may be due to a higher insertion rate, but a low $bmu_1$ *winning* rate.

From the results in Figure 6.2 it is also clear that the number of neurons deleted is scarce throughout the experiments. This means that the data tend to be absorbed always by previously *winning* neurons. This tells us the dataset consists in similar characteristics for normal behaviors (biggest majority of the dataset), therefore they are usually absorbed by neurons that know "well" their characteristics. Also, this tells us that the neurons are tightly connected between in these regions, not allowing the deletion criterion to eliminate them by reason of poor connection of low winning rate.

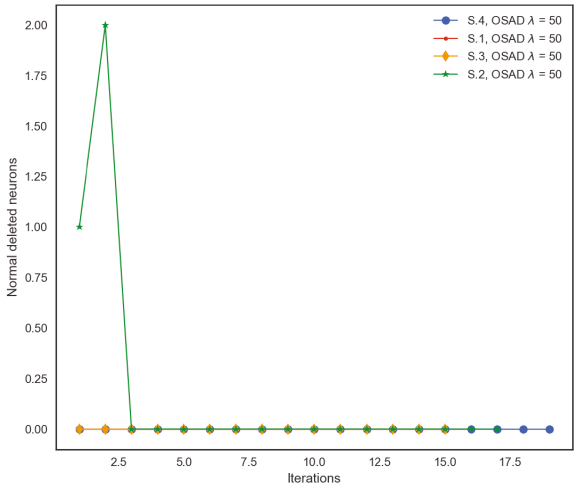
The latter characteristic also benefits from the *reservoir*, experimented against and described in the following section.

### 6.3.4 OSAD Severities and *reservoir*

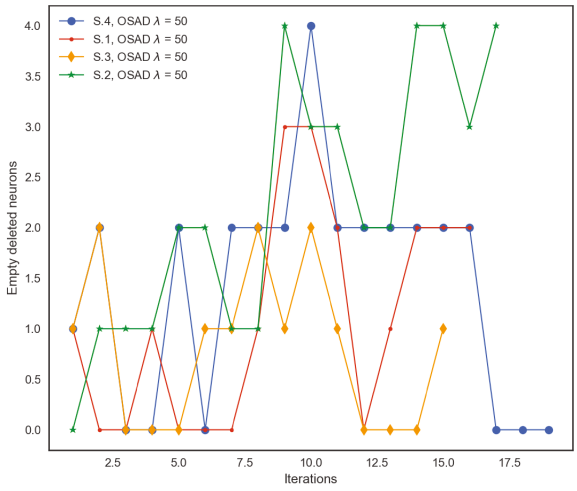
**Severities** As mentioned in Chapter 4, the OSAD makes use of the contextual data given by an expert, resulting in three severity features, namely server, image and network. The algorithm utilizes this information in an adaptive manner by means of thresholds $T_h$, $T_{server}$, $T_{image}$, $T_{network}$, at the moment of inserting and deleting neurons. An example of this calculation is depicted in Figure 6.3 where the red dot represents the percentile, correspondent to the x-axis.

Although these thresholds do not depend on the different subset scenarios, they do rely upon the amount of data inputs the algorithm is fed with.

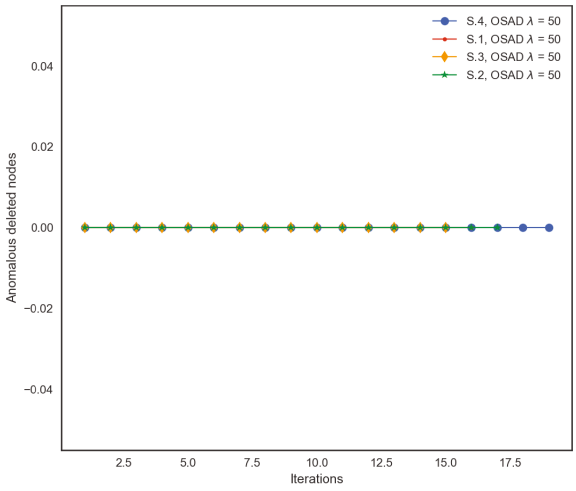
The thresholds are calculated every time an insertion and deletion is about to take place, therefore considering all the data absorbed by the graph at that moment. Consequently, the three severities result in a better representation of the data distribution, after every iteration step or as the algorithm continues learning in time.




(a) *Normal* deleted nodes




(b) Empty deleted nodes




(c) *treacherous* deleted nodes

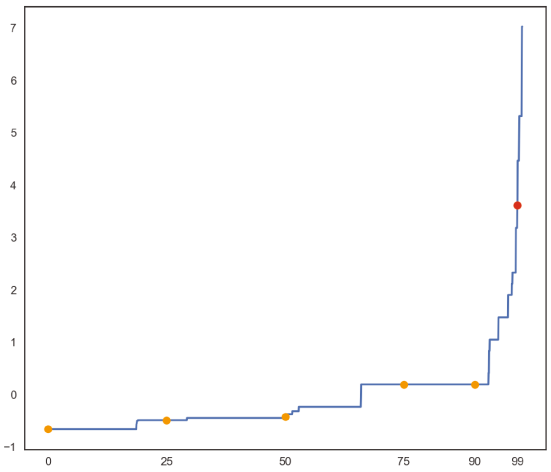
Figure 6.2 – Comparison of deleted nodes for the OSAD algorithm

***Reservoir*** As mentioned in Chapter 4 this module has per objective giving further analysis to the company's experts, along with utilizing the kept neurons as a way of checking new incoming inputs.

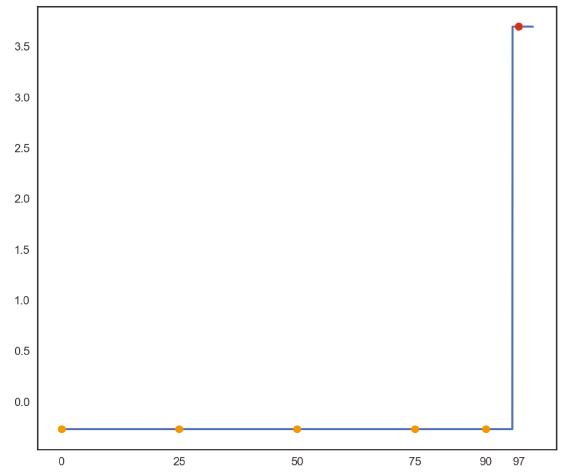
The inclusion of the reservoir serves as an advantage, as seen in Figure 6.4 for the four scenarios, where the number of neurons varies according to the learning evolution of the graph.

The scenarios present different characteristics in the *reservoir*. Figure 6.4 depicts that scenarios 3 and 4 have a different behavior than scenarios 1 and 2. One of the objectives of the *reservoir* is to serve as a temporary buffer, storing *treacherous* neurons for later insertion in the graph. This functionality is not fully utilized by the latter mentioned scenarios as they show fluctuations on the first 5 iterations but then these scenarios increase the number of neurons.

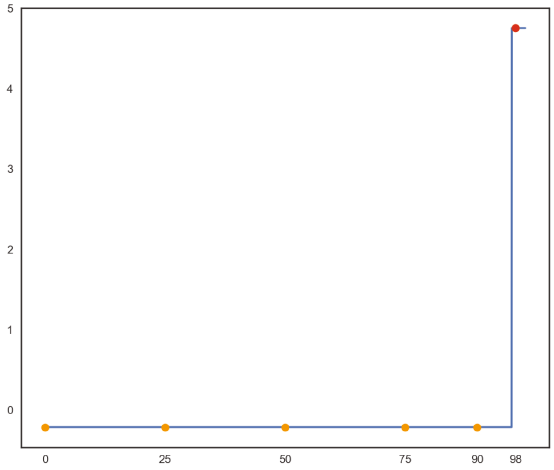
The reason of this behavior may be due to two factors. First, it indicates a poorly-learned graph, where neurons have absorbed both *normal* and *anomalous* input vectors. This means the neurons in the graph are not that specific of the different behaviors, therefore the severities




(a) Severity threshold $T_s$ is the value of percentile 99.97




(b) Severity threshold $T_i$ is the value of percentile 97.85




(c) Severity threshold $T_n$ is the value of percentile 98.60

Figure 6.3 – Adaptive Severity thresholds for 1000 rows of data for first iteration

associated with them are a combination of non-severe and severe values. This characteristic makes new *treacherous* input vectors not finding the BMU, and inserting new outer neurons following the Algorithm 7 and the Algorithm 8 criteria. This is visibly the case of scenario 2, which shows the worst learning evolution in Figure 6.1. Second, this behavior indicates poorly connected regions in the graph.

### 6.3.5 Clustering behavior

In order to establish the detection performance with respect to the model and the different scenarios, *ground-truth* labels have been added to the datasets. Consequently, external metrics have been calculated for the classification towards the unsupervised clustering model. The results can be seen for every iteration in Figure 6.5. Also, Table 6.3 shows the averaged metrics (e.g.,

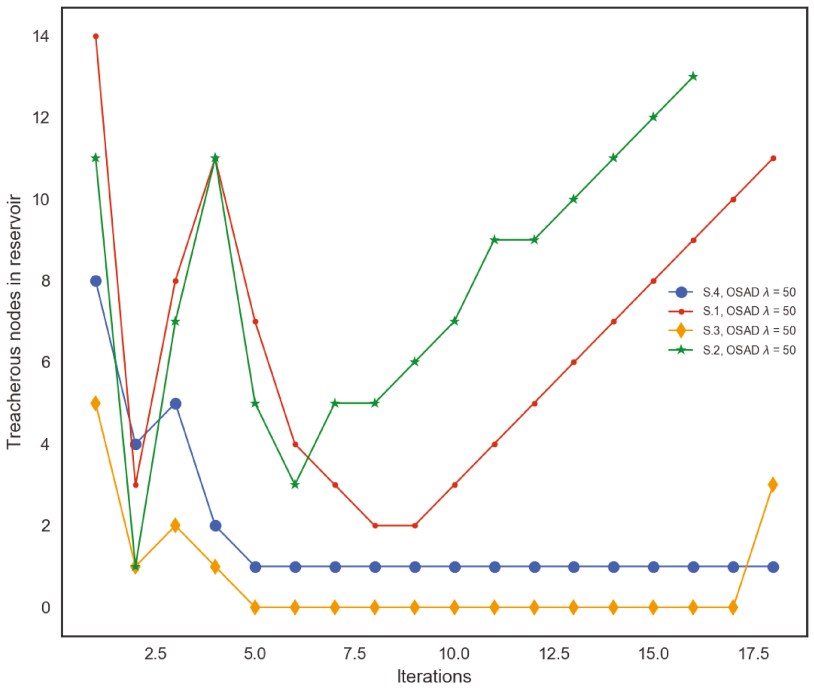



Figure 6.4 – Number of *treacherous* neurons in *reservoir*

F1-score, precision, recall, FPR) and the sum of all the FN, FP, TP and TN.

These results show high values of precision for all studied scenarios. Thus, the model is able to determine when an *anomalous/treacherous* action takes place precisely. However, differences can be seen regarding the recall values, the quotient which considers the FN instead of the FP. For this metric, scenario 2 shows the worst sensitivity towards the proper classification of *anomalous/treacherous* inputs absorbed in the graph. Scenario 1 and 3 show similar results between them, while scenario 4 performs the best.

The intuition for the latter is that scenario 4 has a good ability to find all the *anomalous* cases within input vectors; nevertheless, it also presents the highest FPR. In other words, it tends to consider more neurons as *treacherous*, therefore erring on its prediction.

Further with the analysis, a contextual and sequences attributes comparison can be made in virtue of studying the different performance between the use of contextual data or sequences.

Moreover, the relation whether it is beneficial to use contextual attributes relies upon scenarios 1 and 3. Figure 6.5 depicts that both cases oscillate similarly throughout the iterations, with the exception that scenario 3 results in slightly better precision in its predictions.

Likewise, scenarios 2 and 4 utilize the actions' alphabet as part of their feature subsets. The intuition behind their results is that whenever sequences are present in the set of attributes, they lead the algorithm to misinterpretation. That is the reason these two scenarios have the highest FPR and lower precision values, unlike scenarios 1 and 3.

The explanation for this is that even though the applied $n$-gram lexicographical sequence-to-vector technique considers the letter occurrences, the proposed case scenario only has single lexicographical sequences for the anomalous case. Moreover, the generated *normal* traces also have many lexicographical sequences of length 1, and a few longer sequences (bigger than 1). Thus, *normal* and *anomalous* $n$-grams, are rather binary and similar (e.g., an "L" *normal* se-

quence is no different from an "X" or "T" sequence at the vectorization transformation).

Scenario 4, will then inherit the misconceptions of the sequences, therefore obtaining the highest FPR of all four. The result is also due to the addition of the two Boolean contextual attributes, which are discussed next.

Regarding the benefits of contextual information, via the addition of location and working hours attributes, scenario 3 and 4 have considered a Boolean constant changed to 1 whenever the input vector was labeled as *anomalous*. Although it corresponds to additional data, it provides less information to the algorithm. Even if they are sufficiently correlated with the anomalous, these two additional features do not have much variability (either 0 or 1).

| | S.1 | S.2 | S.3 | S.4 |
|---|---|---|---|---|
| F1-score | 0.637 | 0.271 | 0.616 | 0.803 |
| FPR | 0.004 | 0.037 | 0.001 | 0.476 |
| FNR | 0.575 | 0.843 | 0.584 | 0.391 |
| Precision | 0.963 | 0.936 | 0.991 | 0.940 |
| Recall | 0.504 | 0.159 | 0.461 | 0.752 |
| FN | 2664.000 | 7764.000 | 2627.000 | 1067.000 |
| FP | 76.000 | 95.000 | 17.000 | 108.000 |
| TP | 1968.000 | 1441.000 | 1870.000 | 1659.000 |
| TN | 19556.000 | 9646.000 | 16164.000 | 18253.000 |

Table 6.3 – Averaged external metric results for four scenarios with the OSAD algorithm

Nevertheless, this could be enhanced with new ways of dealing with contextual data, given the fact that the algorithm at the moment of learning, utilizes the Euclidean distance (i.e., the root of a sum of squares). This distance metric is applied for different dimensional vector, i.e., the 34 or above–depending on the scenario, resulting in these two features to play a minor role.

## 6.4 Other experimental Comparisons

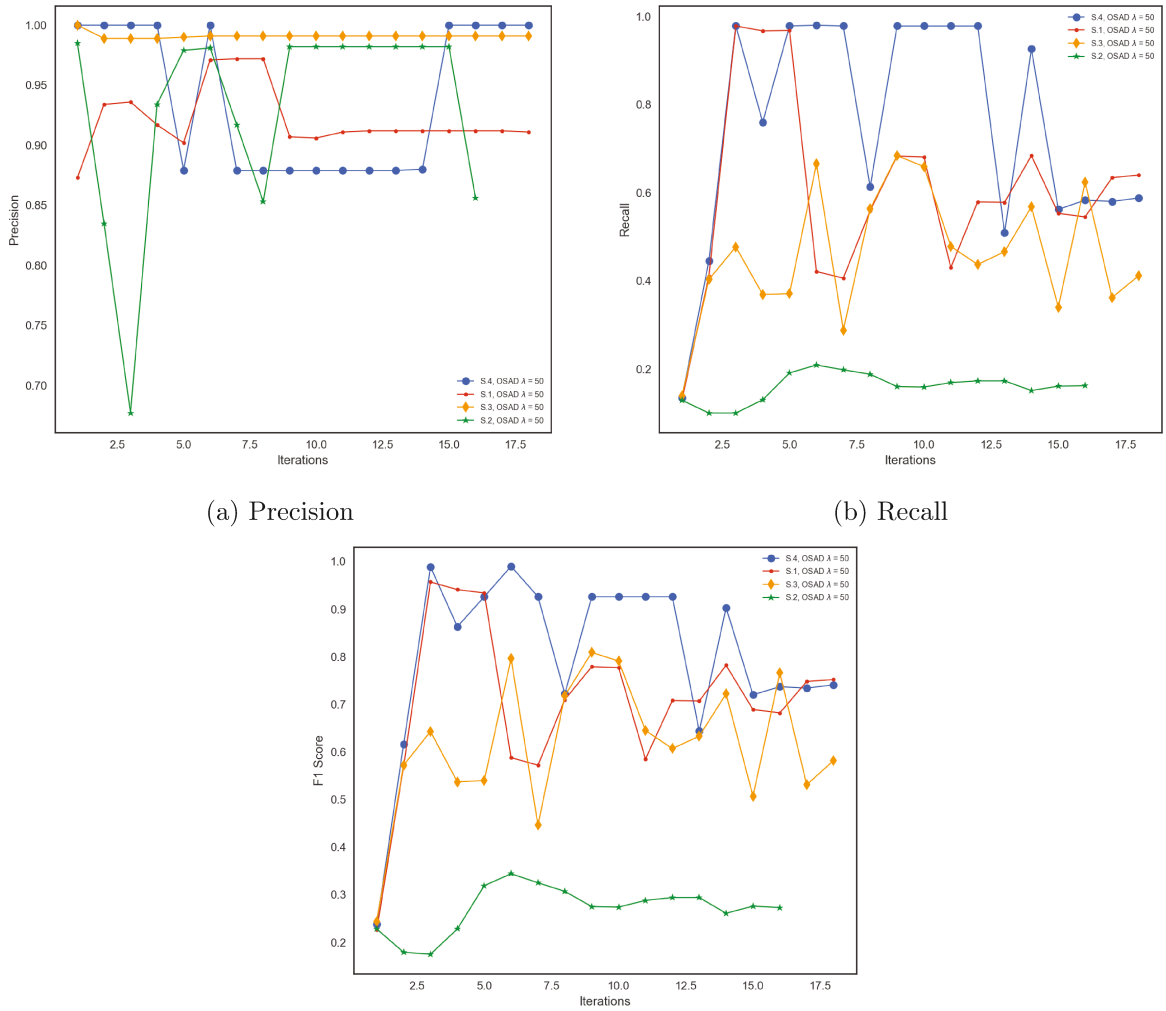
### 6.4.1 Balanced vs unbalanced dataset

We know that whenever dealing with anomaly detection, the general assumption made in the literature and in our work, is the fact of having a big proportion of the data to behave in a way, and having a small portion of the data that follows another conduct.

That being said, for our unsupervised case, there are some disadvantages while working with a balanced dataset. As previously mentioned in Chapter 4, the anomaly detection engine follows a labeling methodology for prediction that relies on the representation of this unbalanced proportion. For this matter, it is counter-intuitive to train a learning algorithm with evenly distributed classes, impairing the labeling phase, based on similarities and sizes.

Nevertheless, the aim of this experiment was also to evaluate the labeling criteria presented in Section 4.3.3.3 given a bigger *anomalous* proportion within the dataset.

To analyze the effects, experiments with two different anomaly/normal ratios were carried. For both settings random number of samples was taken, restricted with the same ratio between *anomalous* and *normal* samples. First scenario, namely *scn*-18 consisted in 448 *normal* and 79 *anomalous* samples giving a ratio of 0.176, while second scenario, namely *scn*-33 consisted in 526 *normal* and 181 *anomalous* samples giving a ratio of 0.334. The experiment considered a number of iterations to also see the evolution and fluctuations.




(a) Precision

(b) Recall

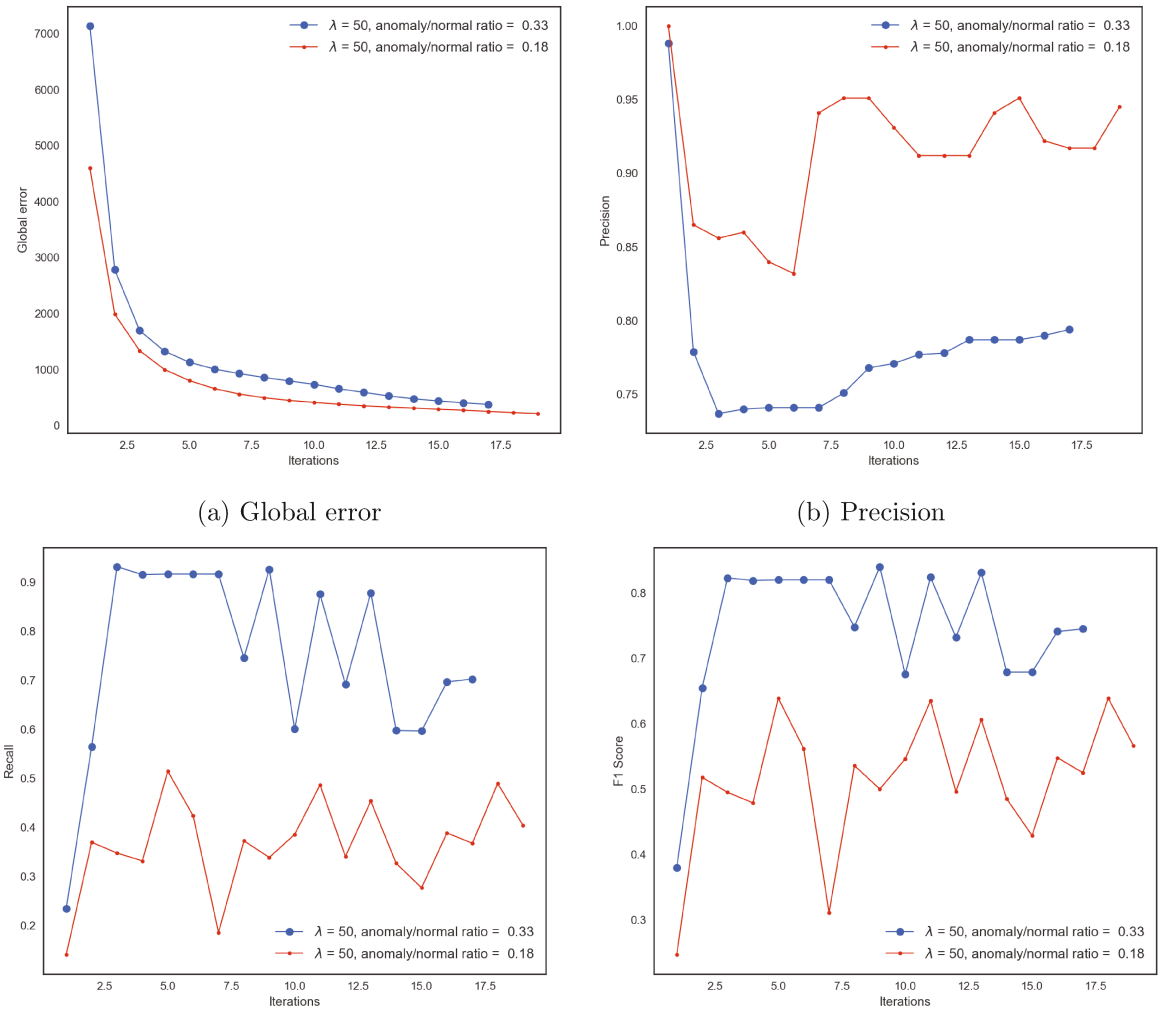
(c) F1-score

Figure 6.5 – External evaluation metrics through labeling for the 4 scenarios

The evaluation metrics, along with the global error throughout the detection procedure can be seen in Figure 6.6. The global error (Figure 6.6a) for both cases shows a good learning performance, where no abrupt increases are seen.

With regards to the evaluation metrics, *scn*-18 has a higher precision and a lower false positive rate than *scn*-33. Thus, the former scenario overlooks the total amount of *treacherous* neurons in the graph, but when it identifies one, it is usually right about that decision. On the contrary, as usually seen in this recall/precision trade-off, the opposite case is taken by *scn*-33, where the tendency is to consider more neurons as *treacherous*, erring in most of its predictions.

Nevertheless, both scenarios present good precision rates, above the 75% (91.3% and 77.8% for *scn*-18 and *scn*-33, respectively), leading to low FPR values (0.62% and 1.2% for *scn*-18 and *scn*-33, respectively).




(a) Global error

(b) Precision

(c) Recall

(d) F1-score

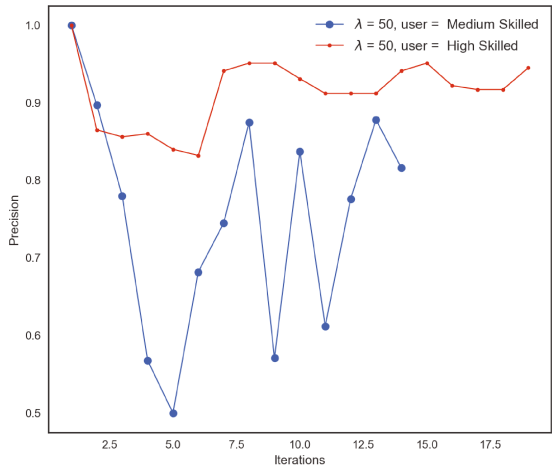
Figure 6.6 – External evaluation metrics for different anomaly/normal ratio with scenario 1

### 6.4.2 Differences between users

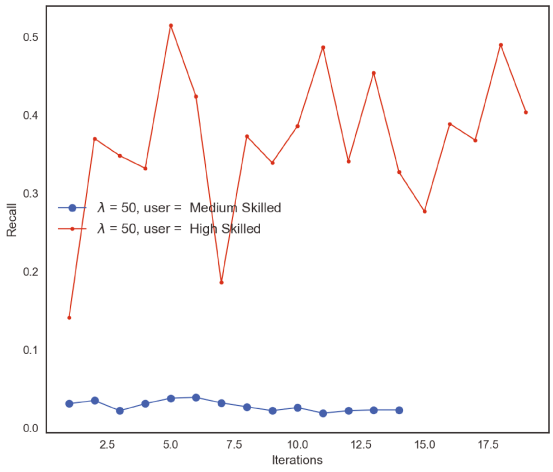
In Section 6.3.1, experiments were carried with different learning rates, also considering different user profiles. An evaluation of two profiles was carried and can be seen in Figure 6.7. The characterization of the two users follows the same configuration of the experimental work derived in Chapter 3.

From these results in Figure 6.7c we can see notorious fluctuations in the detection performance when changing the insider model configuration. This result is primarily relying on the low values from the Recall, as seen in Figure 6.7b.

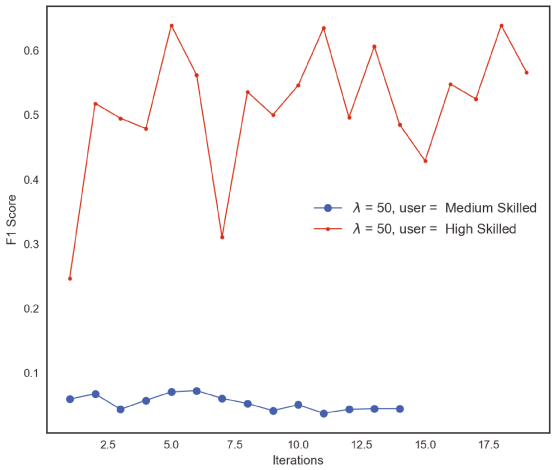
Thus, when aiming for detecting *treacherous* neurons given the medium-skilled user, the framework is able to detect only a small fraction of all the neurons from that category, though when it does, it is usually right, as seen in Figure 6.7a.




(a) Precision




(b) Recall




(c) F1-score

Figure 6.7 – External evaluation metrics for different users with scenario 1

### 6.4.3 Different distance metrics

In Chapter 4, the proposed and compared algorithms rely on comparing the input vector x, $x \in \mathbb{R}^n$ ($n$ features), with the existing neurons, where Euclidean distance is used as the metric.

In virtue of extending the analysis to how the distance metric affects the overall clustering algorithm and its resultant graph, via evaluation metrics, further experiments were carried.

Three distance metrics have been used facing the four detection scenarios, namely Minkowski, Manhattan and the previous Euclidean, as seen in Figure 6.8.

Minkowski distance is the generalized form of different distance measures like Euclidean distance, Manhattan distance. The Minkowski distance is a general formula where $p = 1$ results in Manhattan distance, $p = 2$ results in Euclidean distance. These metrics have been chosen for the benchmark because of the good results presented for a group of clustering techniques with different datasets in [122]. Also, these three distance metrics benefit from having easy time complexity ($O(n)$) [122], where $n$ is the number of dimensions of the input vectors.

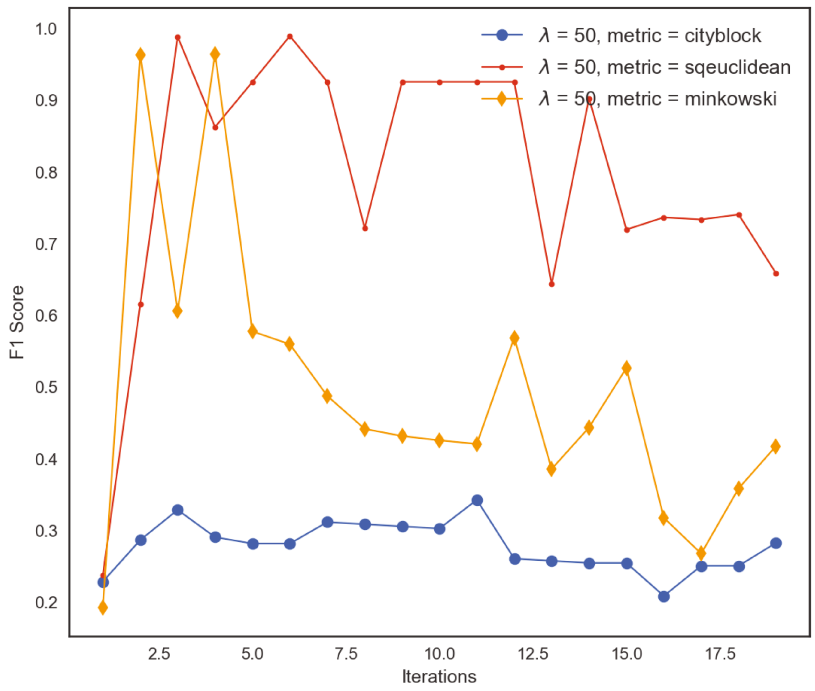



Figure 6.8 – F1-score of Euclidean, Manhattan and Minkowski distance metrics for scenario 4

Results show that the detection framework with the Euclidean distance metric behaves better than the other two. In virtue of these results, the algorithm employs the mentioned distance metric, for its stability and efficiency in the learning process, and due to its accuracy and high F1-score.

## 6.5 Discussion

The present chapter describes different particularities the detection framework depends on, and therefore, should consider.

First, the representation of users' actions. There is an obvious advantage by adopting the action's description within the feature space. This depiction allows classifying the data inputs based not only on the actions' numerical representations under other layers, such as network or operative system, but also under the specific representation of that action in more detailed information. For example, this could allow differentiating large sequences of few actions "A" that all together could resemble to action "B" characteristics. Hence, it comes extremely convenient while trying to cluster these non-easily separable or noisy data.

Nonetheless, given the specificity of the actions to consider, it may be a much more arduous task to keep in maintenance. Moreover, given the evolving scenario of one possible employee's actions, these sets may not be the same at the moment of training and testing. Even in a fully online detection methodology, the continuous update of this information is not a trivial task. Specifically, the deployment's complexity is going to depend on the feature representation. If the feature set is not obliged to be fully numerical, these actions could be treated as such. However, for most of the machine learning and other techniques, numerical feature sets are needed.

Consequently, the applied data representation consisted in $n$-grams. The advantage behind this is because it converts text (actions) into numerical values, which are the value type only plausible for the detection algorithm.

Additionally, experimental results were carried for different representations of sequence features as numerical data, partly analyzed in Chapter 4 (specifically in the Feature Analysis Section 4.3.1) and depicted in Table A.1. These experiments considered not only the $n$-gram representation for $n$ length of 1, 2, and 3, but also the alphabet-to-vector representation.

Thus, considering this relevant factor for detection purposes, requires further investigation. Following that path, preliminary experimental work is being carried, in order to make use of other data representations such as "bag-of-words" or window-based techniques and compare them against the already performed experiments.

Second, the performance of the *reservoir* within the detection framework. The was a noticeable advantage regarding the evaluation metrics, by the proposition of the *reservoir*. All results showed to have good precision and very low FPR, and in some case very good recall values. These results have benefited from the mentioned vault, which keeps for some time, the predicted as *treacherous* neurons.

However, in some of the scenarios, a steady increment of the number of neurons could be seen (e.g., Figure 6.4). This result was the case for a poorly-learned graph, therefore including neurons that held merged *normal* and *treacherous* input vectors.

Hence, new adaptions to this matter are already being considered and evaluated experimentally. Preliminary studies are being held in order to examine a *fading* function, capable of deleting neurons within the *reservoir*, that have not been put back into the algorithm's graph in a period.

Lastly, the detection frameworks' distance metric. In most high dimensional applications the choice of the distance metric is not obvious, and the notion for the calculation of similarity is very heuristical. For example, some distance metrics may be known to have poor accuracy (e.g., Mean Character Difference) and others may behave accurately but with a high computational complexity cost (e.g., Cosine measure and Mahalanobis distance, both $O(3n)$ where $n$ is the number of features, as stated in [122]) for high dimensional data, which is the case for our generated data.

Regarding this matter, additional metrics have been evaluated against, but need further investigation for all the benchmarks mentioned above. Also, further ideas of applying compound criteria of combined distances, is in the research interest.

Finally, by means of pursuing prompt detection, preliminary studies are being done upon different dataset sizes. This factor gives more insights into the amount of time for training in order to reach a stable graph, while also experimenting with different iteration values. Also, it gives crucial information regarding the processing and storage capabilities (e.g., *reservoir*) to consider when aiming to implement the anomaly-based detection framework in production conditions.

## 6.6 Conclusion

In this Chapter, we attempted to benchmark the proposed model of Chapter 4 concerning datasets in consideration of different user profiles, distance metrics, dataset imbalance ratio, among others.

Results show that the OSAD properly clusters the different scenarios, finding topological structures that closely reflect the structure of the input distribution.

Moreover, when evaluated against *ground-truth* classification, it is seen throughout all the experiments, that the results have good decision performance. Also, experiments were carried for two different employees, given from the simulation configuration. It is seen then, that the F1-scores can vary significantly for different profile behaviors. Thus, two conclusions are provided: first, the dataset generation is, in fact, capable of resulting in datasets with different levels of heterogeneity, and second, the detection framework should be further improved to obtain good results for both profiles and moreover, a broader set of profile configurations.

Additionally, the nature of the threats being a rare event makes the possibility of having evenly distributed datasets, a hard task. From both experiments carried within this scope, the detection framework behaved showing low FPR values and a precision rate above the 75%. This result is precisely the benefit of all the adaptive modules of the OSAD algorithm and even more, of the labelling module, which uses contextual data. Hence, even for less unbalanced classes, it is able to use the contextual severity criterion to find the *treacherous* neurons in the graph.

Nonetheless, this can be overcome by data processing techniques such as under-sampling, which should be added to the overall non-unsupervised implementation. Hence for evaluation purposes and techniques benchmarking, we think it is an interesting approach to consider while working with supervised or semi-supervised learning techniques.

Moreover, for insider threat detection, we are usually only interested in the performance of the detector when its FPR is low. Otherwise, too many employees will be falsely suspected of being malicious insiders, and conducting countermeasures for all of them becomes impractical. With this respect, the OSAD algorithm outperforms the compared algorithms, and maintains a low FPR in all evaluated benchmark experiments.

In summary, clear advantages can be found via the utilization of an unsupervised clustering technique, given the fact that obtained results prove it can adequately detect anomalies in different cases.

Chapter 7

# Conclusion and perspectives

**Contents**



## 7.1 Main results

Insider threats present many challenges, given that they consist in actions of humans and not machines, and detection techniques will inevitably incorporate characteristics of human behavior as opposed to the more purely technical domains of network data generation. Moreover, providing data for insider threat research presents an unusual challenge for many reasons.

In order to detect this threat, there are two options: those requiring use or collection of real user data, and those using synthetic data. In order to collect real data, a CSC must directly monitor and record the behavior and actions of its employees. However, confidentiality and privacy concerns create barriers to the collection and manipulation of such data for research, development, and evaluation.

Accordingly, the works organized and presented in this document describe a solution to detect the threat mentioned above, utilizing an unsupervised approach, an adaptive technique towards evolving behaviors, and an extensible framework for deriving datasets regarding this threat, as well as monitoring its most important attributes.

The first contribution of this work aimed at designing and developing a simulation engine, capable of tackling these issues and resulting in datasets for detection purposes. It is able to derive different scenarios from different aspects. On the one hand, given a probabilistic approach, the developed simulation engine allows modeling a user relative to the technical skills they have towards cloud assets, and also depending on their psychological predisposition to perform an *anomalous* or treacherous act. Also, following a higher optic, it allows the simulation of a single or multiple users in a company, all of them executing real requests to a cloud implementation. This approach is a crucial advantage at the moment of deriving different enterprise scenarios, in the lack of real data, granting adaptability to the specific needs of industrial corporations.

The second and main contribution represents the base of the whole detection framework for the efficient detection of the insider threat. It corresponds to a novel anomaly-based detection

framework, and it is framed under the application and extension of a machine learning technique, adapted for the anomaly detection of the insider threat. This module is designed to represent past, present and evolving behaviors of employee's actions towards the cloud assets of a company. Structurally, it is composed of two main algorithms: a training phase and online phase. The design has been carried in such a way that the training phase can determine whenever the trained algorithm has stabilized, by internal clustering metrics such as the modularity. The algorithm then has learned the normal and abnormal behaviors of the user after capturing data in an elapsed period, and by iterating itself through that data until the resultant graph topology does not show significant fluctuations.

Additionally, the framework proposed a continuous adjustment of community detection, where the clusters are reconstructed based on the adaptation against new incoming data, proving to be capable of identifying new emerging *treacherous* communities of neurons/clusters, together with *normal* communities, which also evolve, grow, split-up or become more behavior-specific in time.

The latter procedure works in conjunction with the proposed labeling criteria, which has been evaluated extensively against different experiments in Section 6. Results show the proposed detection accurately identifies *anomalous* behaviors, under different benchmark studies. Furthermore, compared to other clustering techniques presented in Chapter 4, the proposed solution also shows better comparative FPR and F1-score results at the moment of identifying *treacherous* actions.

This contribution also consisted in presenting novelties in the sections of data representation for further detection. The detection framework combined user-based data (i.e., literal strings representing actions), network-based and cloud-based attributes, along with contextual knowledge information.

Regarding the benchmark results in Chapter 6, for the analysis of the 4 scenarios and considering the low-FPR premise, it is better to work with contextual data (FPR for scenario 3 presents the lowest at 0.001%) rather than with sequences. Accordingly, the best performance is held by scenarios 3 (presenting the lowest at 0.001%) and 1. They both show very similar characteristics, low FPR while having good precision at the moment identifying all the *treacherous* neurons in the graph.

Hence, in virtue of an efficient trade-off between the number of features to monitor, it is advised to pursue the set given by scenario 3, and moreover, if efficiency and implementation is a relevant issue, the feature set of scenario 1 is a good candidate, which holds good values for F1-score, while only utilizing network traces and IaaS-related features.

Concerning the different data attributes of study, the essential features for the *anomalous* and treacherous detection were classified by their contribution with random forests. This well-known machine learning method is often utilized for feature selection. However, it requires *ground-truth* labels, namely, it is a supervised technique. Since the proposed work aims at achieving results in an unsupervised manner, the methodology to assess this estimation was using the proposed labeling criteria given by the framework, and then concluding on the relevant features for each of the *normal*, *treacherous* and *unknown* classes.

Results from that analysis concluded that regarding the comparative importance between the different subsets, the network-based features contributed with more information, due to their variability and richer numerical representation. Relevant features of this group are `Total Pkts`, `Response Time`, `DL Data Vol`, `UL Data Vol`, `Octet Stream Type` and `Session Time`. These features could be then selected as a priority subset, if needed to increase computational efficiency.

IaaS-based features also proved to supply complementary information, such as `Server Shutoff`

`Mean`, `Server Delete Mean` and `Server Stop Mean`. This is logical, given the fact that the *anomalous* activities concerned these three actions.

Moreover, when looking into the contextual attributes' relevance, obtained results showed these attributes paid a crucial role in contributing to the identification of *treacherous* traces, both in: (1) internal features fed by the OSAD algorithm, and (2) using the proposed labeling criteria.

Sequences on the other hand, whenever present, add information since they are specifying the real action to which the trace corresponds. Results, in this case, show the importance given to the overall dataset, depends mainly on two aspects. First, on the action sequences derived by the insider threat simulation, especially their level of randomness at the profile configuration stage. The configuration relates to their length and the occurrence of every action within them. Second and most importantly, their importance depends on the way they are represented (e.g., numerically, as strings).

Moreover, the proposed $n$-gram string-to-vector representation of the actions does not seem to provide vital information. The reason depends on the factors mentioned above, which for the generated datasets, may not have been too heterogeneous and fundamentally different between them (i.e., normal generated actions were slightly longer in string-length than the anomalous) therefore, at the moment of vectorizing them numerically, they did not differ much. Nevertheless, the extensible approach derived from the simulation in Chapter 3 allows further investigation in this matter, through the generation of longer and more complex sequences of actions.

All conclusions considered above give essential insights of the relevant attributes to consider whenever wanting to detect behaviors that diverge from the typical usage of cloud resources, considering the suggestions at the moment of implementing monitoring sensors, prepossessing the data, and detecting correspondingly.

Another aspect of the proposed solution is the concept of *reservoir*, which is a concept previously applied (e.g., [50] but for different purposes than anomaly detection). In the present work, this component has been designed to allow keeping historical track of all the decisions of the users during their requests towards the cloud. At the same time, it effectively tackles the issue of belated detection characteristics this threat has in particular. All benchmark experiments were also analyses concerning this component, in Chapter 6. Results depicted the expected behavior, in most cases, i.e., the module serves as a temporary vault for *treacherous* neurons. Consequently, for most cases, results depict increments of the number of neurons, followed by decrements. Correspondingly, it effectively permits the company to audit and make decisions promptly when anomalies are detected.

The third contribution of this work represents the validation process of the proposed framework. By means of characterizing different scenarios, we make use of various subsets to prove the anomalies' classification through different datasets. As a consequence, through the adaptive learning, the experimental work shows that the proposed schema benefits from several factors. First of all, it makes every neuron more stable and also more specific to its particular prototype. This means that after a long constant time period, every neuron will have clustered more similar input vectors, decreasing the accumulated local error and improving the topological structure of the graph $\mathcal{G}$. Moreover, this means the heterogeneous behavior a user can have in a company, is properly described by a set of these neurons, therefore tackling the characterization of evolving execution tasks by the employees. Furthermore, by virtue of the contextual added features, the framework is able to label certain behaviors for future decision making.

The fourth contribution of this work is formed by the implementation of the aforementioned monitoring approach, composed by two principal components: a DPI for network traffic provided

by MMT, along with the proposed implementation for monitoring relevant IaaS features, capable of detecting critical situations. By virtue of the latter module, the framework is able to monitor an extensive set of attributes, that automatically retrieve the IaaS information from the cloud. This also has a flexible approach in mind, by being able to obtain information on not only existing assets, but also capable of incorporating new assets, as part of cloud principal characteristics. Additionally, it has an extensive modular design in mind, by means of monitoring not only the relevant attributes for the portrayed threat scenarios, but also additional future actions the simulation could consider executing.

## 7.2 Perspectives

The proposed model in Chapter 3 considers the fundamental characteristics to illustrate a user model: different technical capabilities, psychological propensity (motive) and the opportunity to execute an attack. On this behalf, it is considered that adding an inter-psychological factor would enrich the simulation. This factor would be useful to characterize more complex psychological and social dynamics. Following the latter, there is much more to explore as to enterprise schema/hierarchies and how social interactions could also be affected by them. These attributes may consider the representation of different organization logistics, among others. Also, it would permit to model insider threat scenarios such as collusion or social engineering. All these characteristics may extend the `Context` entity of the proposed user model. For example, some organizations may have a *crossfunctional* business structure, where the employees are expected to interact across functional areas, therefore, the *normal* behavior is broader and could be more heterogeneous.

Concerning the evaluation of the proper functioning of the simulation tool, further enhancements can also be made regarding its analysis through richer techniques, such as statistical control. This method was mentioned in the discussion during the defense of this work, and presents benefits at the moment of working with a simulation tool that generates different datasets every time, therefore having an inherent statistical nature. Accordingly, the use of a method like this, would enhance the evaluation of the different properties following a criteria based in acceptable ranges with intervals of trust, also provided at the configuration step of the simulation; instead of rigid comparatives such as fix thresholds.

Also, by incorporating the knowledge of the consequences after pursuing a malicious act, the user model comportment could enrich themselves by the use of reinforcement learning theory. This machine learning technique falls under the idea of a more complex insider model that allows reasoning about the risk associated and its consequences, as well as richer conducts in the context of personal motives, capabilities, and opportunity.

Finally, further studies in this area may contribute to the derivation of an organizational tool that benefits from studying their employees and analyze risky behaviors regarding vulnerability scans.

Concerning the specific components of the proposed detection framework, more experiments could be done towards the optimization of the *reservoir*. Its capacity is limited and in some cases could accumulate too many neurons, such as an abrupt group of anomalies arrival or high-severity behaviors. For that concern, it is necessary to implement enriched policies in order to maintain relevant information only. For instance, in the case of the examples mentioned above, the *reservoir* could be capable of executing a different procedure, such as different deletion for buffer optimization, along with a possible alert notification.

Regarding anomaly detectors' functionality, future explorations can be done on how feature

selection can play a role in the cyber-security domain, and further studies should be carried to obtain more efficient ways of data representation in continually evolving threat patterns. This research would also be advantageous since it results in algorithms that require less storage volume, and lower demand for resources concerning processing.

Lastly, additional relevant features from knowledge in the domain may be studied, helping via contextual information, for more efficient learning engine in the detection framework. Moreover, contextual information was used internally by the detection algorithm. Nevertheless, further studies regarding the use of these features separately by means of categorizing the incoming vectors in a precedent step may also optimize the algorithm's performance by having fewer dimensions. Such would be the case of the use for the proposed severities and the policy based contextual information location and working hours.

Regarding data collection, enhancements could be made, building new security monitoring methodologies in order to enforce availability and reliability. Consequently, various challenges need to be considered such as, for example, storing and synchronizing in distributed environments that have different Availability Zones (AZs) as well as multiple tenants, distributed logically and geographically.

Moreover, concerning monitoring implementation and also for the detection engine, a new design of a parallel version of the proposed method would take advantage of data-intensive computing platforms, such as Apache Spark. One way to do this would be to parallelize independent parts of the data and process them in parallel while sharing the same graph of neurons. Another alternative would be to design algorithms to adequately split and distribute the graph of neurons on multiple machines running in parallel.

Lastly, the proposed solution can be evaluated against more insider threat case scenarios. For this matter and although non-cloud related, the CERT dataset [17] can be considered, i.e., the known and used (e.g., [49], [52], [127]) insider threat dataset from the past years. Therefore, further work towards the evaluation of detection performance with this dataset is of major interest.

# Bibliography

[1] A. S. Abed, T. C. Clancy, and D. S. Levy, "Applying bag of system calls for anomalous behavior detection of applications in linux containers", in *2015 IEEE Globecom Workshops, San Diego, CA, USA, December 6-10, 2015*, IEEE, 2015, pp. 1–5. DOI: 10.1109/GLOCOMW.2015.7414047. [Online]. Available: https://doi.org/10.1109/GLOCOMW.2015.7414047.

[2] C. C. Aggarwal, *Outlier Analysis*, 2nd. Springer Publishing Company, Incorporated, 2016.

[3] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams", in *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, ser. VLDB '03, Berlin, Germany: VLDB Endowment, 2003, pp. 81–92, ISBN: 0-12-722442-4. [Online]. Available: http://dl.acm.org/citation.cfm?id=1315451.1315460.

[4] Akamai, "Akamai's state of the internet/Security Q3 2015 report", Tech. Rep., 2015.

[5] S. S. Alarifi and S. D. Wolthusen, "Detecting anomalies in iaas environments through virtual machine host system call analysis", in *7th International Conference for Internet Technology and Secured Transactions, ICITST 2012, London, United Kingdom, December 10-12, 2012*, N. Savage, S. E. Assad, and C. A. Shoniregun, Eds., IEEE, 2012, pp. 211–218. [Online]. Available: http://ieeexplore.ieee.org/document/6470945/.

[6] A. Angelopoulou, A. Psarrou, and J. García Rodríguez, "A growing neural gas algorithm with applications in hand modelling and tracking", in *Advances in Computational Intelligence*, J. Cabestany, I. Rojas, and G. Joya, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 236–243, ISBN: 978-3-642-21498-1.

[7] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores", in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, ser. CCS '07, 2007.

[8] Z. A. Baig and F. Binbeshr, "Controlled Virtual Resource Access to Mitigate Economic Denial of Sustainability (EDoS) Attacks against Cloud Infrastructures", *2013 International Conference on Cloud Computing and Big Data (CloudCom-Asia)*, vol. 00, pp. 346–353, 2013.

[9] W. Baker, A. Hutton, D. Hylender, C. Novak, C. Porter, B. Sartin, P. Tippett, J. Andrew Valentine, T. Bosschert, E. Brohm, C. Chang, R. Dormido, and K. a. Eric Gentry, "2009 Data Breach Investigations Report", Tech. Rep., 2009.

[10] A. Bates, B. Mood, J. Pletcher, H. Pruse, M. Valafar, and K. Butler, "Detecting co-residency with active traffic analysis techniques", in *CCSW '12*, New York, New York, USA: ACM Press, 2012, pp. 1–12.

[11] V. D. Blondel, J.-l. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks", 2008.

[12] B. Böse, B. Avasarala, S. Tirthapura, Y. Y. Chung, and D. Steiner, "Detecting insider threats using radish: a system for real-time anomaly detection in heterogeneous data streams", *IEEE Systems Journal*, vol. 11, no. 2, pp. 471–482, 2017, ISSN: 1932-8184. DOI: 10.1109/JSYST.2016.2558507.

[13] M.-R. Bouguelia, S. Nowaczyk, and A. H. Payberah, "An adaptive algorithm for anomaly and novelty detection in evolving data streams", *Data Mining and Knowledge Discovery*, 2018, ISSN: 1573-756X. DOI: 10.1007/s10618-018-0571-0. [Online]. Available: https://doi.org/10.1007/s10618-018-0571-0.

[14] L. Brian, G. Joshua, R. Mitch, and W. Kurt C., "Generating test data for insider threat detectors", *JoWUA*, vol. 5, no. 2, pp. 80–94, 2014.

[15] U. of California. (2018). Uc irvine machine learning repository, [Online]. Available: https://archive.ics.uci.edu/ml/index.php (visited on 05/16/2018).

[16] J. Cao, B. Yu, F. Dong, X. Zhu, and S. Xu, "Entropy-based denial of service attack detection in cloud data center", in *2014 Second International Conference on Advanced Cloud and Big Data*, Nov. 2014, pp. 201–207.

[17] S. E. I. Carnegie Mellon University. (2016). Insider Threat Test Dataset, [Online]. Available: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=508099 (visited on 05/16/2018).

[18] P. Carvallo, A. R. Cavalli, and N. Kushik, "A study of threat detection systems andâ techniques in the cloud", in *Risks and Security of Internet and Systems*, N. Cuppens, F. Cuppens, J.-L. Lanet, A. Legay, and J. Garcia-Alfaro, Eds., Cham: Springer International Publishing, 2018, pp. 140–155, ISBN: 978-3-319-76687-4.

[19] P. Carvallo, A. R. Cavalli, and W. Mallouli, "A platform for security monitoring of multi-cloud applications", in *Perspectives of System Informatics*, A. K. Petrenko and A. Voronkov, Eds., Cham: Springer International Publishing, 2018, pp. 59–71, ISBN: 978-3-319-74313-4.

[20] P. Carvallo, A. R. Cavalli, W. Mallouli, and E. Rios, "Multi-cloud applications security monitoring", in *Green, Pervasive, and Cloud Computing*, M. H. A. Au, A. Castiglione, K.-K. R. Choo, F. Palmieri, and K.-C. Li, Eds., Cham: Springer International Publishing, 2017, pp. 748–758, ISBN: 978-3-319-57186-7.

[21] P. Carvallo, A. R. Cavalli, and N. Kushik, "Automatic derivation and validation of a cloud dataset for insider threat detection", in *ICSOFT 2017 : 12th International Conference on Software Technologies*, Madrid, Spain: Scitepress, Jul. 2017, pp. 480–487. DOI: 10.5220/0006480904800487. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01583232.

[22] B. Caswell, J. C. Foster, R. Russell, J. Beale, and J. Posluns, *Snort 2.0 Intrusion Detection*. Syngress Publishing, 2003, ISBN: 1931836744.

[23] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: a survey", *ACM Comput. Surv.*, vol. 41, no. 3, 15:1–15:58, Jul. 2009, ISSN: 0360-0300. DOI: 10.1145/1541880.1541882. [Online]. Available: http://doi.acm.org/10.1145/1541880.1541882.

[24] H.-H. Chou and S.-D. Wang, "An adaptive network intrusion detection approach for the cloud environment", in *2015 International Carnahan Conference on Security Technology (ICCST)*, IEEE, 2015, pp. 1–6.

[25] Cisco, "Cisco 2017 Annual Cybersecurity Report", Tech. Rep., 2017.

[26] W. R. Claycomb and A. Nicoll, "Insider threats to cloud computing: directions for new research challenges", in *2012 IEEE 36th Annual Computer Software and Applications Conference*, Jul. 2012, pp. 387–394.

[27] Cloud Security Alliance CSA, "The Treacherous 12 - Cloud Computing Top Threats in 2016", 2016.

[28] M. Collins, M. Theis, R. Trzeciak, J. Strozer, J. Clark, D. Costa, T. Cassidy, M. Albrethsen, and A. Moore, "Common sense guide to mitigating insider threats", Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2016-TR-015, 2016. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=484738.

[29] D. Costa, M. Collins, S. J. Perl, M. Albrethsen, G. Silowash, and D. Spooner, "An ontology for insider threat indicators: development and application", in *Proceedings of the Ninth Conference on Semantic Technology for Intelligence, Defense, and Security, Fairfax VA, USA, November 18-21, 2014.*, 2014, pp. 48–53.

[30] G. Creech and J. Hu, "Generation of a new ids test dataset: time to retire the kdd collection", in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, Apr. 2013, pp. 4487–4492.

[31] Crowd Research Partners. (2018). Insider threat report, [Online]. Available: http://crowdresearchpartners.com/wp-content/uploads/2017/07/Insider-Threat-Report-2018.pdf (visited on 05/20/2018).

[32] C. S. A. CSA, "The Notorious Nine: Cloud Computing Top Threats in 2013", 2013.

[33] A. Duncan, S. Creese, and M. Goldsmith, "An overview of insider attacks in cloud computing", *Concurrency and Computation: Practice and Experience*, vol. 27, no. 12, pp. 2964–2981, 2015, ISSN: 1532-0634.

[34] W. Eberle, J. Graves, and L. Holder, "Insider threat detection using a graph-based approach", *Journal of Applied Security Research*, vol. 6, no. 1, pp. 32–81, 2010. DOI: 10.1080/19361610.2011.529413.

[35] H. Eldardiry, E. Bart, J. Liu, J. Hanley, B. Price, and O. Brdiczka, "Multi-domain information fusion for insider threat detection", in *2013 IEEE Security and Privacy Workshops*, May 2013, pp. 45–51. DOI: 10.1109/SPW.2013.14.

[36] ENISA, "ENISA Threat Landscape 2015", Tech. Rep., Jan. 2016.

[37] J. M. Epstein, *Agent_ Zero:Toward Neurocognitive Foundations for Generative Social Science*. Princeton University Press, 2014.

[38] Ernst and Young, "Managing insider threat: A holistic approach to dealing with risk from within", Tech. Rep., 2016.

[39] C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic Provable Data Possession", *ACM Transactions on Information and System Security (TISSEC)*, vol. 17, no. 4, 2015.

[40] F. Farahmand and E. H. Spafford, "Understanding insiders: an analysis of risk-taking behavior", *Information Systems Frontiers*, vol. 15, no. 1, pp. 5–15, Mar. 2013, ISSN: 1572-9419. DOI: 10.1007/s10796-010-9265-x. [Online]. Available: https://doi.org/10.1007/s10796-010-9265-x.

[41] D. A. B. Fernandes, L. F. B. Soares, J. V. Gomes, M. M. Freire, and P. R. M. Inácio, "Security issues in cloud environments: a survey", *Int. J. Inf. Secur.*, 2014.

[42] R. T. Fielding and J. Reschke, *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*, RFC 7230, Jun. 2014. DOI: 10.17487/RFC7230. [Online]. Available: https://rfc-editor.org/rfc/rfc7230.txt.

[43] B. Fritzke, "A growing neural gas network learns topologies", in *Proceedings of the 7th International Conference on Neural Information Processing Systems*, ser. NIPS'94, Denver, Colorado: MIT Press, 1994, pp. 625–632. [Online]. Available: http://dl.acm.org/citation.cfm?id=2998687.2998765.

[44] S. Furao and O. Hasegawa, "An incremental network for on-line unsupervised classification and topology learning", *Neural Networks*, vol. 19, no. 1, pp. 90–106, 2006, ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2005.04.006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608005000845.

[45] A. Gamachchi, L. Sun, and S. Boztas, "Graph based framework for malicious insider threat detection", in *50th Hawaii International Conference on System Sciences, HICSS 2017, Hilton Waikoloa Village, Hawaii, USA, January 4-7, 2017*, 2017. [Online]. Available: http://aisel.aisnet.org/hicss-50/eg/insider_threat/2.

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

[47] J. GarcíaRodriguez and J. M. G. Chamizo, "Surveillance and human computer interaction applications of self-growing models", *Applied Soft Computing*, vol. 11, no. 7, pp. 4413–4431, 2011, Soft Computing for Information System Security, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2011.02.007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494611000676.

[48] G. Garkoti, S. K. Peddoju, and R. Balasubramanian, "Detection of Insider Attacks in Cloud Based e-Healthcare Environment", in *2014 International Conference on Information Technology (ICIT)*, IEEE, 2014, pp. 195–200.

[49] G. Gavai, K. Sricharan, D. Gunning, J. Hanley, M. Singhal, and R. Rolleston, "Supervised and unsupervised methods to detect insider threat from enterprise social and online activity data", *JoWUA*, vol. 6, pp. 47–63, 2015.

[50] M. Ghesmoune, M. Lebbah, and H. Azzag, "A new growing neural gas for clustering data streams", *Neural Netw.*, vol. 78, no. C, pp. 36–50, Jun. 2016, ISSN: 0893-6080. DOI: 10.1016/j.neunet.2016.02.003. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2016.02.003.

[51] I. A. Gheyas and A. E. Abdallah, "Detection and prediction of insider threats to cyber security: a systematic literature review and meta-analysis", *Big Data Analytics*, vol. 1, no. 1, p. 6, 2016. DOI: 10.1186/s41044-016-0006-0. [Online]. Available: https://doi.org/10.1186/s41044-016-0006-0.

[52] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data", *PLoS ONE*, vol. 11, no. 4, pp. 1–31, Apr. 2016. DOI: 10.1371/journal.pone.0152173. [Online]. Available: https://www.goldiges.de/publications/Comparison_Unsupervised_Anomaly_Detection.pdf.

[53] F. L. Greitzer, M. Imran, J. Purl, E. T. Axelrad, Y. M. Leong, D. E. Becker, K. B. Laskey, and P. J. Sticha, "Developing an ontology for individual and organizational sociotechnical indicators of insider threat risk", in *STIDS*, 2016.

[54] S. Gupta, P. Kumar, A. Sardana, and A. Abraham, "A fingerprinting system calls approach for intrusion detection in a cloud environment", in *2012 Fourth International Conference on Computational Aspects of Social Networks (CASoN)*, IEEE, 2012, pp. 309–314.

[55] O. Hamdi, M. Mbaye, and F. Krief, "A cloud-based architecture for network attack signature learning", in *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*, IEEE, 2015.

[56] J. Han, W. Li, Z. Su, L. Zhao, and W. Deng, "Community detection by label propagation with compression offlow", *The European Physical Journal B*, vol. 89, no. 12, p. 272, 2016, ISSN: 1434-6036. DOI: 10.1140/epjb/e2016-70264-6. [Online]. Available: https://doi.org/10.1140/epjb/e2016-70264-6.

[57] K. Hashizume, D. G. Rosado, E. Fernández-Medina, and E. B. Fernandez, "An analysis of security issues for cloud computing", *Journal of Internet Services and Applications*, vol. 4, no. 1, p. 5, 2013, ISSN: 1869-0238.

[58] S.-Y. Huang, N. Suri, and Y. Huang, "Event Pattern Discovery on IDS Traces of Cloud Services", in *2014 IEEE International Conference on Big Data and Cloud Computing (BdCloud)*, IEEE, 2014, pp. 25–32.

[59] T. Huang, Y. Zhu, Y. Wu, S. Bressan, and G. Dobbie, "Anomaly detection and identification scheme for VM live migration in cloud infrastructure", *Future Generation Computer Systems*, vol. 56, pp. 736–745, Mar. 2016.

[60] H. Idrissi, S. E. Hajji, M. Ennahbaoui, E. M. Souidi, and E. M. Souidi, "Mobile Agents with Cryptographic Traces For Intrusion Detection in the Cloud Computing", *Procedia Computer Science*, vol. 73, 2015.

[61] N. Jeyanthi, N. C. S. N. Iyengar, P. C. M. Kumar, and A. Kannammal, "An Enhanced Entropy Approach to Detect and Prevent DDoS in Cloud Environment.", *IJCNIS 5(2)*, vol. 5, no. 2, 2013.

[62] A. Jones and D. Ashenden, *Risk Management for Computer Security: Protecting Your Network & Information Assets*. Newton, MA, USA: Butterworth-Heinemann, 2005, ISBN: 0750677953.

[63] M. Jouad, S. Diouani, H. Houmani, and A. Zaki, "Security challenges in intrusion detection", in *2015 International Conference on Cloud Technologies and Applications (CloudTech)*, IEEE, 2015, pp. 1–11.

[64] D. Kahneman and A. Tversky, "Prospect Theory: An Analysis of Decision under Risk", *Econometrica*, vol. 47, no. 2, pp. 263–291, Mar. 1979. [Online]. Available: https://ideas.repec.org/a/ecm/emetrp/v47y1979i2p263-91.html.

[65] M. Kandias, A. Mylonas, N. Virvilis, M. Theoharidou, and D. Gritzalis, "An insider threat prediction model", in *Trust, Privacy and Security in Digital Business: 7th International Conference, TrustBus 2010, Bilbao, Spain, August 30-31, 2010. Proceedings*, S. Katsikas, J. Lopez, and M. Soriano, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 26–37, ISBN: 978-3-642-15152-1.

[66] M. Kandias, N. Virvilis, and D. Gritzalis, "The insider threat in cloud computing", in *Critical Information Infrastructure Security: 6th International Workshop, CRITIS 2011, Lucerne, Switzerland, September 8-9, 2011, Revised Selected Papers*, S. Bologna, B. Hämmerli, D. Gritzalis, and S. Wolthusen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 93–103.

[67] D.-K. Kang, D. Fuller, and V. Honavar, "Learning classifiers for misuse and anomaly detection using a bag of system calls representation", in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, Jun. 2005, pp. 118–125. DOI: 10.1109/IAW.2005.1495942.

[68] G. Katz, Y. Elovici, and B. Shapira, "CoBAn: A context based model for data leakage prevention", *Information Sciences: an International Journal*, vol. 262, pp. 137–158, 2014.

[69] S. G. Kene and D. P. Theng, "A review on intrusion detection techniques for cloud computing and security challenges", in *2015 2nd International Conference on Electronics and Communication Systems (ICECS)*, IEEE, 2015, pp. 227–232.

[70] H. A. Kholidy and F. Baiardi, "CIDD: A Cloud Intrusion Detection Dataset for Cloud Computing and Masquerade Attacks", in *2012 Ninth International Conference on Information Technology: New Generations (ITNG)*, IEEE, 2012, pp. 397–402.

[71] M. T. Khorshed, A. B. M. S. Ali, and S. A. Wasimi, "A survey on gaps, threat remediation challenges and some thoughts for proactive attack detection in cloud computing", *Future Generation Computer Systems*, vol. 28, no. 6, pp. 833–851, Jun. 2012.

[72] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *CoRR*, 2014.

[73] T. Kohonen, M. R. Schroeder, and T. S. Huang, Eds., *Self-Organizing Maps*, 3rd. Berlin, Heidelberg: Springer-Verlag, 2001, ISBN: 3540679219.

[74] E. Kowalski, D. Cappelli, and A. P. Moore, "Insider Threat Study: Illicit Cyber Activity in the Information Technology and Telecommunications Sector", Software Engineering Institute, Tech. Rep., Jan. 2008.

[75] N. Kumar, V. Katta, H. Mishra, and H. Garg, "Detection of Data Leakage in Cloud Computing Environment", in *2014 International Conference on Computational Intelligence and Communication Networks (CICN)*, IEEE, 2014, pp. 803–807.

[76] S. Lee, G. Kim, and S. Kim, "Self-adaptive and dynamic clustering for online anomaly detection", *Expert Systems with Applications*, vol. 38, no. 12, pp. 14 891–14 898, 2011.

[77] P. A. Legg, O. Buckley, M. Goldsmith, and S. Creese, "Automated insider threat detection system using user and role-based profile assessment", *IEEE Systems Journal*, vol. 11, no. 2, pp. 503–512, 2017, ISSN: 1932-8184. DOI: 10.1109/JSYST.2015.2438442.

[78] P. Legg, N. Moffat, J. R. C. Nurse, I. Agrafiotis, M. Goldsmith, and S. Creese, "Towards a conceptual model and reasoning structure for insider threat detection", in *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications 4(4*, 2013.

[79] K. Leung and C. Leckie, "Unsupervised anomaly detection in network intrusion detection using clusters", in *Proceedings of the Twenty-eighth Australasian Conference on Computer Science - Volume 38*, ser. ACSC '05, Newcastle, Australia: Australian Computer Society, Inc., 2005, pp. 333–342, ISBN: 1-920-68220-1. [Online]. Available: `http://dl.acm.org/citation.cfm?id=1082161.1082198`.

[80] Y.-H. Li, Y.-R. Tzeng, and F. Yu, "VISO: Characterizing Malicious Behaviors of Virtual Machines with Unsupervised Clustering", in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, IEEE, 2015.

[81] Lincon Laboratory MIT. (2017). Darpa intrusion detection evaluation, [Online]. Available: `https://www.ll.mit.edu/ideval/data/index.html`.

[82] L. Liu, O. D. Vel, Q. L. Han, J. Zhang, and Y. Xiang, "Detecting and preventing cyber insider threats: a survey", *IEEE Communications Surveys Tutorials*, vol. 20, no. 2, pp. 1397–1417, 2018. DOI: `10.1109/COMST.2018.2800740`.

[83] A. K. Marnerides, Petros, P. Spachos, P. Chatzimisios, and A. U. Mauthe, "Malware detection in the cloud under Ensemble Empirical Mode Decomposition", in *2015 International Conference on Computing, Networking and Communications (ICNC)*, IEEE, 2015, pp. 82–88.

[84] A. K. Marnerides, N.-h. Shirazi, D. Hutchison, S. Simpson, M. Watson, and A. Mauthe, "Assessing the impact of intra-cloud live migration on anomaly detection", in *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, 2014.

[85] T. M. Martinetz and K. J. Schulten, "A "neural gas" network learns topologies", in *Proceedings of the International Conference on Artificial Neural Networks 1991* (Espoo, Finland), T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, Eds., Amsterdam; New York: North-Holland, 1991, pp. 397–402.

[86] P. M. Mell and T. Grance, "SP 800-145. The NIST Definition of Cloud Computing", Gaithersburg, MD, United States, Tech. Rep., 2011.

[87] C. N. Modi and D. Patel, "A novel hybrid-network intrusion detection system (H-NIDS) in cloud computing", *2013 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, pp. 23–30, 2013.

[88] C. Modi, D. R. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in Cloud.", *JNCA*, vol. 36, no. 1, pp. 42–57, 2013.

[89] MongoDB. (2018). MongoDB, [Online]. Available: `https://www.mongodb.com/` (visited on 05/16/2018).

[90] A. P. Moore, D. M. Cappelli, T. C. Caron, E. D. Shaw, D. Spooner, and R. F. Trzeciak, "A preliminary model of insider theft of intellectual property", *JoWUA*, vol. 2, no. 1, pp. 28–49, 2011. [Online]. Available: `http://isyou.info/jowua/papers/jowua-v2n1-2.pdf`.

[91] P. Moriano, J. Pendleton, S. Rich, and L. J. Camp, "Insider threat event detection in user-system interactions", in *Proceedings of the 2017 International Workshop on Managing Insider Security Threats*, ser. MIST '17, Dallas, Texas, USA: ACM, 2017, pp. 1–12, ISBN: 978-1-4503-5177-5. DOI: 10.1145/3139923.3139928. [Online]. Available: http://doi.acm.org/10.1145/3139923.3139928.

[92] K. Murphy, *Machine Learning: A Probabilistic Perspective*, ser. Adaptive computation and machine learning. MIT Press, 2012, ISBN: 9780262018029.

[93] MUSA, *MUSA H2020 project*, http://www.musa-project.eu/, (Retrieved May 2017), 2017.

[94] M. E. J. Newman, "Modularity and community structure in networks", *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006. DOI: 10.1073/pnas.0601602103. eprint: http://www.pnas.org/content/103/23/8577.full.pdf+html. [Online]. Available: http://www.pnas.org/content/103/23/8577.abstract.

[95] L. Nkosi, P. Tarwireyi, and M. O. Adigun, "Insider threat detection model for the cloud", in *2013 Information Security for South Africa*, Aug. 2013, pp. 1–8.

[96] O. I. de Normalización, *ISO/IEC 27005: Information technology-Security techniques - Information security risk management*. ISO, 2008. [Online]. Available: https://books.google.fr/books?id=K1HbZwEACAAJ.

[97] J. R. C. Nurse, O. Buckley, P. A. Legg, M. Goldsmith, S. Creese, G. R. T. Wright, and M. Whitty, "Understanding insider threat: a framework for characterising attacks", in *Proceedings of the 2014 IEEE Security and Privacy Workshops*, ser. SPW '14, Washington, DC, USA: IEEE Computer Society, 2014, pp. 214–228, ISBN: 978-1-4799-5103-1. DOI: 10.1109/SPW.2014.38. [Online]. Available: http://dx.doi.org/10.1109/SPW.2014.38.

[98] OpenStack. (2018). OpenStack software, [Online]. Available: https://www.openstack.org/ (visited on 05/16/2018).

[99] Openstack. (2018). Openstack Server Statuses description, [Online]. Available: https://developer.openstack.org/api-guide/compute/server_concepts.html (visited on 05/16/2018).

[100] O. Osanaiye, K.-K. R. Choo, and M. Dlodlo, "Distributed denial of service (DDoS) resilience in cloud: Review and conceptual cloud DDoS mitigation framework", *Journal of Network and Computer Applications*, Jan. 2016.

[101] N. Pandeeswari and G. Kumar, "Anomaly Detection System in Cloud Environment Using Fuzzy Clustering Based ANN", *Mobile Netw Appl*, pp. 1–12, 2015.

[102] D. B. Parker, *Fighting Computer Crime: A New Framework for Protecting Information*. New York, NY, USA: John Wiley & Sons, Inc., 1998, ISBN: 0-471-16378-3.

[103] P. Parveen, J. Evans, B. Thuraisingham, K. W. Hamlen, and L. Khan, "Insider threat detection using stream mining and graph mining", in *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, Oct. 2011, pp. 1102–1110. DOI: 10.1109/PASSAT/SocialCom.2011.211.

[104] A. Patel, M. Taghavi, K. Bakhtiyari, and J. Celestino Júnior, "An intrusion detection and prevention system in cloud computing: A systematic review", *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 25–41, Jan. 2013.

[105] M. A. F. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "Review: a review of novelty detection", *Signal Process.*, vol. 99, pp. 215–249, 2014, ISSN: 0165-1684. DOI: 10.1016/j.sigpro.2013.12.026. [Online]. Available: http://dx.doi.org/10.1016/j.sigpro.2013.12.026.

[106] L. Portnoy, E. Eskin, and S. Stolfo, "Intrusion detection with unlabeled data using clustering", in *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*, 2001, pp. 5–8.

[107] RabbitMQ. (2018). Rabbitmq, [Online]. Available: https://www.rabbitmq.com/ (visited on 05/16/2018).

[108] T. Rashid, I. Agrafiotis, and J. R. Nurse, "A new take on detecting insider threats: exploring the use of hidden markov models", in *Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats*, ser. MIST '16, Vienna, Austria: ACM, 2016, pp. 47–56, ISBN: 978-1-4503-4571-2. DOI: 10.1145/2995959.2995964. [Online]. Available: http://doi.acm.org/10.1145/2995959.2995964.

[109] R. A. Rescorla and A. R. Wagner, "A theory of Pavlovian conditioning: variations on the effectiveness of reinforcement and non-reinforcement", in *Classical conditioning II: Current research and theory*, A. H. Black and W. F. Prokasy, Eds., New York: Appleton-Century-Crofts, 1972, pp. 64–99.

[110] D. G. Rosado, R. Gómez, D. Mellado, and E. Fernández-Medina, "Security Analysis in the Migration to Cloud Environments", *Future Internet*, vol. 4, no. 4, pp. 469–487, Dec. 2012.

[111] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis", *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987, ISSN: 0377-0427. DOI: https://doi.org/10.1016/0377-0427(87)90125-7. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0377042787901257.

[112] K. Roy Sarkar, "Assessing insider threats to information security using technical, behavioural and organisational measures", *Inf. Secur. Tech. Rep.*, vol. 15, no. 3, pp. 112–133, Aug. 2010, ISSN: 1363-4127. DOI: 10.1016/j.istr.2010.11.002. [Online]. Available: http://dx.doi.org/10.1016/j.istr.2010.11.002.

[113] M. B. Salem and S. J. Stolfo, "Modeling user search behavior for masquerade detection", in *Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection*, ser. RAID'11, Menlo Park, CA: Springer-Verlag, 2011, pp. 181–200, ISBN: 978-3-642-23643-3.

[114] E. Schultz, "A framework for understanding and predicting insider attacks", *Comput. Secur.*, vol. 21, no. 6, pp. 526–531, Oct. 2002, ISSN: 0167-4048. DOI: 10.1016/S0167-4048(02)01009-X. [Online]. Available: http://dx.doi.org/10.1016/S0167-4048(02)01009-X.

[115] D. Sculley, "Web-scale k-means clustering", in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10, Raleigh, North Carolina, USA: ACM, 2010, pp. 1177–1178, ISBN: 978-1-60558-799-8.

[116] T. E. Senator, H. G. Goldberg, A. Memory, W. T. Young, B. Rees, R. Pierce, D. Huang, M. Reardon, D. A. Bader, E. Chow, I. Essa, J. Jones, V. Bettadapura, D. H. Chau, O. Green, O. Kaya, A. Zakrzewska, E. Briscoe, R. I. L. Mappus, R. McColl, L. Weiss, T. G. Dietterich, A. Fern, W. K. Wong, S. Das, A. Emmott, J. Irvine, J.-Y. Lee, D. Koutra, C. Faloutsos, D. Corkill, L. Friedland, A. Gentzel, and D. Jensen, "Detecting insider threats in a real corporate database of computer usage activity", in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '13, Chicago, Illinois, USA: ACM, 2013, pp. 1393–1401, ISBN: 978-1-4503-2174-7. DOI: 10.1145/2487575.2488213. [Online]. Available: http://doi.acm.org/10.1145/2487575.2488213.

[117] P. Shamsolmoali, M. A. Alam, and R. Biswas, "C2DF: High Rate DDOS filtering method in Cloud Computing", *International Journal of Computer Network and Information Security*, vol. 6, no. 9, pp. 43–50, Aug. 2014.

[118] P. Sharma, R. Sharma, E. S. Pilli, and A. K. Mishra, "A Detection Algorithm for DoS Attack in the Cloud Environment.", *COMPUTE*, pp. 107–110, 2015.

[119] E. D. Shaw, "The role of behavioral research and profiling in malicious cyber insider investigations", *Digit. Investig.*, vol. 3, no. 1, pp. 20–31, Mar. 2006, ISSN: 1742-2876.

[120] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection.", *Computers & Security*, 2012.

[121] S. N. Shirazi, S. Simpson, A. Gouglidis, A. Mauthe, and D. Hutchison, "Anomaly detection in the cloud using data density", in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, 2016, pp. 616–623.

[122] A. S. Shirkhorshidi, S. Aghabozorgi, and T. Y. Wah, "A comparison study on similarity and dissimilarity measures in clustering continuous data", *PLOS ONE*, vol. 10, no. 12, pp. 1–20, Dec. 2015. DOI: 10.1371/journal.pone.0144059. [Online]. Available: https://doi.org/10.1371/journal.pone.0144059.

[123] J. A. Sokolowski, C. M. Banks, and T. J. Dover, "An agent-based approach to modeling insider threat", *Computational and Mathematical Organization Theory*, vol. 22, no. 3, pp. 273–287, Sep. 2016, ISSN: 1572-9346. DOI: 10.1007/s10588-016-9220-6. [Online]. Available: https://doi.org/10.1007/s10588-016-9220-6.

[124] Y. Song, M. B. Salem, S. Hershkop, and S. J. Stolfo, "System level user behavior biometrics using fisher features and gaussian mixture models", in *2013 IEEE Security and Privacy Workshops*, May 2013, pp. 52–59. DOI: 10.1109/SPW.2013.33.

[125] S. V. Stehman, "Selecting and interpreting measures of thematic classification accuracy", *Remote Sensing of Environment*, vol. 62, no. 1, pp. 77–89, 1997, ISSN: 0034-4257. DOI: https://doi.org/10.1016/S0034-4257(97)00083-7. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0034425797000837.

[126] Q. Sun, H. Liu, and T. Harada, "Online growing neural gas for anomaly detection in changing surveillance scenes", *Pattern Recognition*, vol. 64, pp. 187–201, 2017, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2016.09.016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320316302771.

[127] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, "Deep learning for unsupervised insider threat detection in structured cybersecurity data streams", *CoRR*, vol. abs/1710.00811, 2016.

[128] UNSW, Australian Defense Force Academy. (2017). Adfa ids datasets, [Online]. Available: `https://www.unsw.adfa.edu.au/australian-centre-for-cyber-security/cybersecurity/ADFA-IDS-Datasets/`.

[129] L. M. Vaquero, L. Rodero-Merino, and D. Morán, "Locking the sky: a survey on IaaS cloud security.", *Computing*, vol. 91, no. 1, 2011.

[130] S. M. Varghese and K. P. Jacob, "Process profiling using frequencies of system calls", in *Proceedings of the The Second International Conference on Availability, Reliability and Security, ARES 2007, The International Dependability Conference - Bridging Theory and Practice, April 10-13 2007, Vienna, Austria*, IEEE Computer Society, 2007, pp. 473–479. DOI: `10.1109/ARES.2007.116`. [Online]. Available: `https://doi.org/10.1109/ARES.2007.116`.

[131] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer, "Taxonomy and Survey of Collaborative Intrusion Detection.", *CSUR*, vol. 47, no. 4, pp. 55–33, 2015.

[132] D. Viejo, J. Rodríguez, M. Cazorla, D. Gil, and M. Johnsson, "Using gng to improve 3d feature extraction-application to 6dof egomotion", vol. 32, pp. 138–46, Feb. 2012.

[133] VirtualBox. (2018). VirtualBox software, [Online]. Available: `https://www.virtualbox.org/` (visited on 05/16/2018).

[134] M. R. Watson, N.-h. Shirazi, A. K. Marnerides, A. Mauthe, and D. Hutchison, "Malware Detection in Cloud Computing Infrastructures.", *TDSC*, vol. 13, no. 2, pp. 192–205, 2016.

[135] B. Wehbi, E. M. de Oca, and M. Bourdelles, "Events-based security monitoring using mmt tool", in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, Apr. 2012, pp. 860–863. DOI: `10.1109/ICST.2012.188`.

[136] S. Widup, M. Spitler, D. Hylender, and G. Bassett, "2018 verizon data breach investigations report", Tech. Rep., Apr. 2018.

[137] M. Xie and J. Hu, "Evaluating host-based anomaly detection systems: a preliminary analysis of adfa-ld", in *2013 6th International Congress on Image and Signal Processing (CISP)*, vol. 03, Dec. 2013, pp. 1711–1716. DOI: `10.1109/CISP.2013.6743952`.

[138] Z. Xing, J. Pei, and E. Keogh, "A brief survey on sequence classification", *SIGKDD Explor. Newsl.*, vol. 12, no. 1, pp. 40–48, Nov. 2010, ISSN: 1931-0145. DOI: `10.1145/1882471.1882478`. [Online]. Available: `http://doi.acm.org/10.1145/1882471.1882478`.

[139] W. Xiong, H. Hu, N. Xiong, L. T. Yang, W.-C. Peng, X. Wang, and Y. Qu, "Anomaly secure detection methods by analyzing dynamic characteristics of the network traffic in cloud communications", *Information Sciences*, vol. 258, pp. 403–415, Feb. 2014.

[140] Q. Yaseen, Q. Althebyan, B. Panda, and Y. Jararweh, "Mitigating insider threat in cloud relational databases", *Security and Communication Networks*, 2016.

[141] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, "Beehive: large-scale log analysis for detecting suspicious activity in enterprise networks", in *Proceedings of the 29th Annual Computer Security Applications Conference*, ser. ACSAC '13, New Orleans, Louisiana, USA: ACM, 2013, pp. 199–208, ISBN: 978-1-4503-2015-3. DOI: 10.1145/2523649.2523670. [Online]. Available: http://doi.acm.org/10.1145/2523649.2523670.

[142] W. T. Young, H. G. Goldberg, A. Memory, J. F. Sartain, and T. E. Senator, "Use of domain knowledge to detect insider threats in computer activities", in *2013 IEEE Security and Privacy Workshops*, May 2013, pp. 60–67. DOI: 10.1109/SPW.2013.32.

[143] W. T. Young, A. Memory, H. G. Goldberg, and T. E. Senator, "Detecting unknown insider threat scenarios", in *2014 IEEE Security and Privacy Workshops*, May 2014, pp. 277–288. DOI: 10.1109/SPW.2014.42.

[144] n. S. Yu, S. Yu, n. X. Gui, X. Gui, n. J. Lin, and J. Lin, "An approach with two-stage mode to detect cache-based side channel attacks", in *2013 International Conference on Information Networking (ICOIN)*, IEEE, 2013, pp. 186–191.

[145] W. Yu, P. Moulema, G. Xu, and Z. Chen, "A cloud computing based architecture for cyber security situation awareness", in *2013 IEEE Conference on Communications and Network Security (CNS)*, IEEE, 2013, pp. 488–492.

[146] M. Zbakh, K. Elmahdi, R. Cherkaoui, and S. Enniari, "A multi-criteria analysis of intrusion detection architectures in cloud environments", in *2015 International Conference on Cloud Technologies and Applications (CloudTech)*, IEEE, 2015, pp. 1–9.

[147] X. Zhang, C. Furtlehner, and M. Sebag, "Data streaming with affinity propagation", in *Machine Learning and Knowledge Discovery in Databases*, W. Daelemans, B. Goethals, and K. Morik, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 628–643, ISBN: 978-3-540-87481-2.

Appendix A

# Dataset features

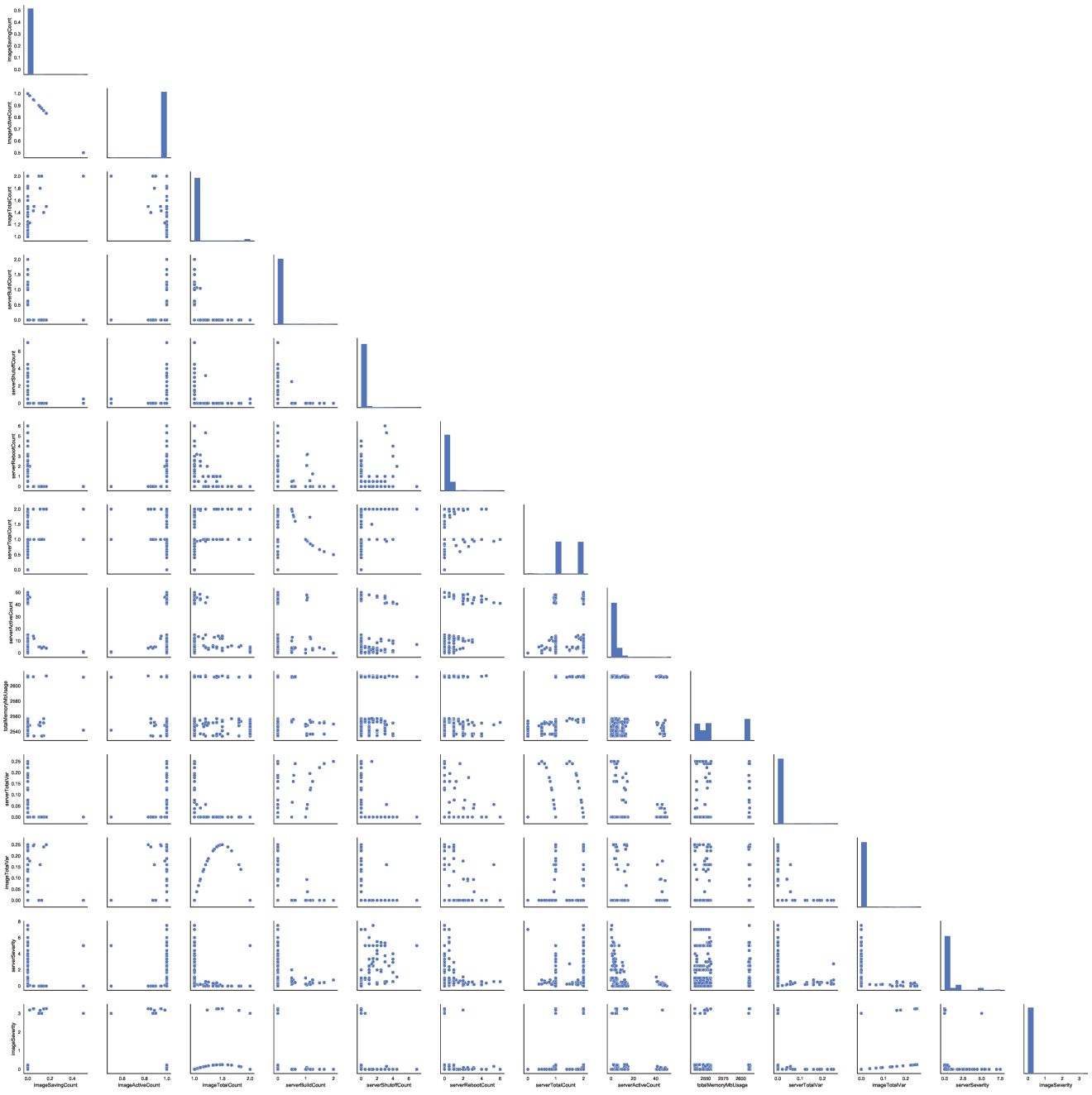



Figure A.1 – Pair-plot for IaaS-based features

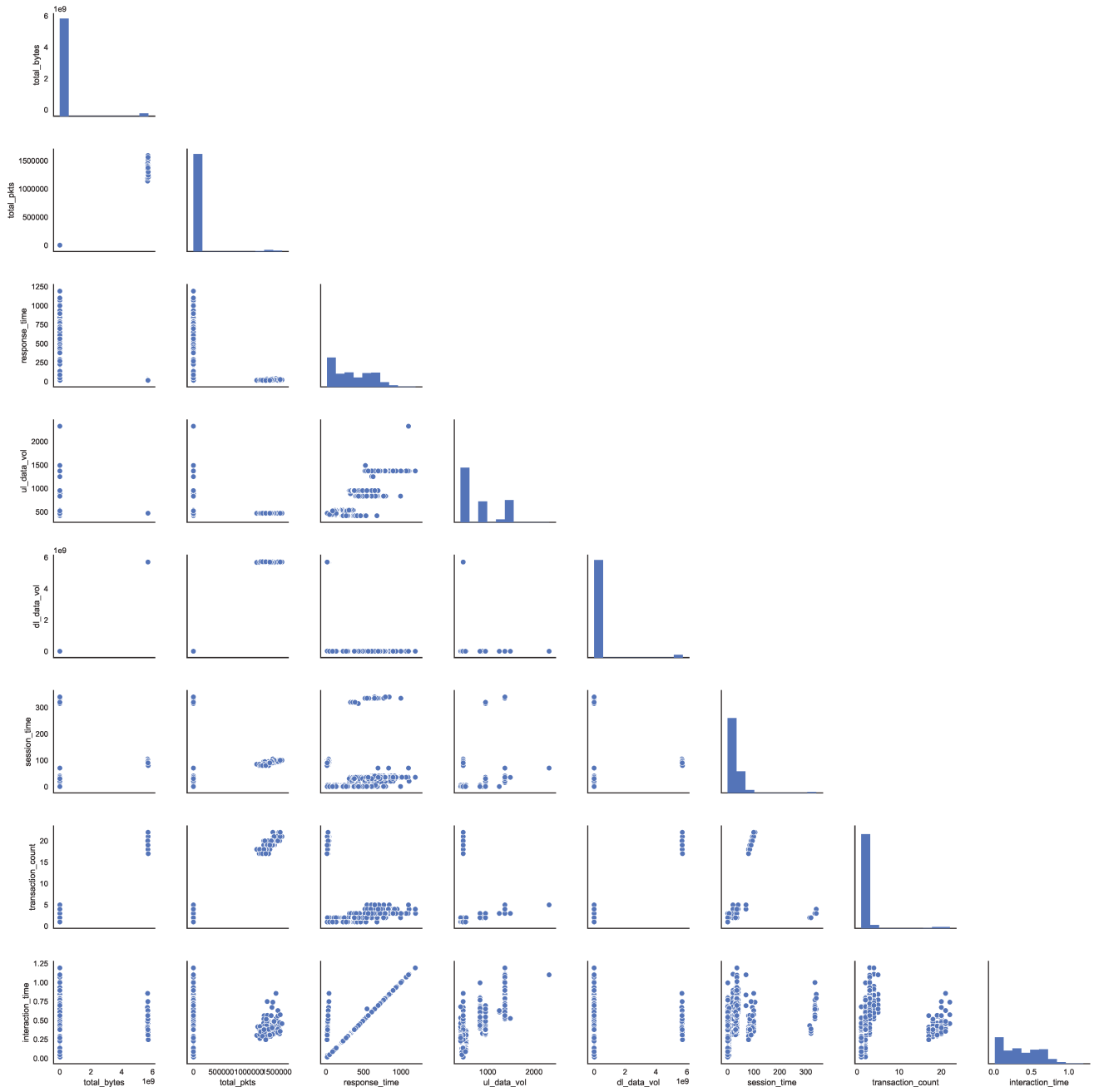



Figure A.2 – Pair-plot for Network-based features

| Scenario | Time | Internal | | Labeling | | | | External | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Modularity | Silhouette | Coms. found | Normal | Threat | Unknown | ARI | NMI | F1 | Accuracy | FPR |
| Cloud Admin High Alphabet Seq Only | 34.603316 | 0.834946 | 0.631 | 13 | 1 | 0 | 12 | 0.237659 | 0.325467 | 0.472 | 0.770 | 0.256 |
| Cloud Admin High Alphabet Seq and other Feats | 54.830397 | 0.818913 | -0.036 | 17 | 12 | 0 | 5 | 0.264090 | 0.058151 | 0.371 | 0.884 | 0.057 |
| Cloud Admin Medium Alphabet Seq Only | 46.414830 | 0.809578 | 0.566 | 11 | 2 | 0 | 9 | 0.048926 | 0.074710 | 0.096 | 0.769 | 0.233 |
| Cloud Admin Medium Alphabet Seq and other Feats | 118.933960 | 0.859154 | 0.237 | 20 | 9 | 0 | 11 | 0.019910 | 0.016907 | 0.059 | 0.710 | 0.289 |
| Cloud Admin High 1-gram Seq-only | 41.656105 | 0.813539 | 0.537 | 11 | 3 | 0 | 8 | 0.530819 | 0.244589 | 0.664 | 0.901 | 0.109 |
| Cloud Admin High 2-gram Seq-only | 35.361977 | 0.825814 | 0.779 | 11 | 5 | 0 | 6 | -0.013859 | 0.121068 | 0.123 | 0.600 | 0.364 |
| Cloud Admin High 3-gram Seq-only | 29.207140 | 0.745466 | 0.513 | 8 | 5 | 0 | 3 | 0.703185 | 0.228799 | 0.761 | 0.958 | 0.001 |
| Cloud Admin High 1-gram Seq-feats | 60.447913 | 0.841771 | 0.129 | 15 | 7 | 0 | 8 | 0.001458 | 0.084937 | 0.181 | 0.532 | 0.465 |
| Cloud Admin High 2-gram Seq-feats | 55.331043 | 0.855454 | 0.426 | 19 | 7 | 0 | 12 | 0.073439 | 0.066765 | 0.298 | 0.669 | 0.332 |
| Cloud Admin High 3-gram Seq-feats | 65.185473 | 0.875944 | 0.297 | 21 | 4 | 0 | 17 | -0.022123 | 0.084752 | 0.230 | 0.462 | 0.577 |
| Cloud Admin Medium 1-gram Seq-only | 26.534031 | 0.750503 | 0.864 | 9 | 4 | 0 | 5 | 0.881494 | 0.047153 | 0.885 | 0.997 | 1.000 |
| Cloud Admin Medium 2-gram Seq-only | 25.847943 | 0.753429 | 0.875 | 10 | 3 | 0 | 7 | 0.043974 | 0.032048 | 0.088 | 0.762 | 0.240 |
| Cloud Admin Medium 3-gram Seq-only | 30.556918 | 0.748077 | 0.630 | 10 | 3 | 0 | 7 | 0.038691 | 0.035276 | 0.079 | 0.776 | 0.223 |
| Cloud Admin Medium 1-gram Seq-feats | 89.450220 | 0.844221 | 0.057 | 19 | 10 | 0 | 9 | 0.008669 | 0.019387 | 0.040 | 0.745 | 0.249 |
| Cloud Admin Medium 2-gram Seq-feats | 94.517919 | 0.842157 | 0.406 | 15 | 10 | 0 | 5 | 0.009375 | 0.010454 | 0.038 | 0.863 | 0.127 |
| Cloud Admin Medium 3-gram Seq-feats | 105.978900 | 0.882283 | 0.378 | 18 | 8 | 0 | 10 | 0.044174 | 0.016188 | 0.089 | 0.757 | 0.245 |
| Cloud Admin High 1-gram Seq-only | 24.502653 | 0.722063 | 0.652 | 8 | 2 | 0 | 6 | 0.181339 | 0.301477 | 0.404 | 0.750 | 0.261 |
| Cloud Admin High 2-gram Seq-only | 27.673261 | 0.736586 | 0.499 | 9 | 3 | 0 | 6 | 0.127958 | 0.170443 | 0.329 | 0.739 | 0.249 |
| Cloud Admin High 3-gram Seq-only | 26.197885 | 0.764599 | 0.737 | 10 | 6 | 0 | 4 | 0.227217 | 0.121573 | 0.421 | 0.797 | 0.191 |
| Cloud Admin High 1-gram Seq-feats | 57.601754 | 0.822315 | 0.413 | 14 | 5 | 0 | 9 | 0.191802 | 0.098439 | 0.353 | 0.815 | 0.149 |
| Cloud Admin High 2-gram Seq-feats | 65.644418 | 0.827478 | 0.146 | 15 | 7 | 0 | 8 | 0.061508 | 0.072392 | 0.261 | 0.694 | 0.280 |
| Cloud Admin High 3-gram Seq-feats | 56.945146 | 0.861759 | 0.388 | 16 | 8 | 0 | 8 | 0.102926 | 0.071671 | 0.276 | 0.758 | 0.208 |
| Cloud Admin Medium 1-gram Seq-only | 34.623877 | 0.770876 | 0.921 | 10 | 5 | 0 | 5 | 0.825137 | 0.060661 | 0.831 | 0.996 | 1.000 |
| Cloud Admin Medium 2-gram Seq-only | 36.335605 | 0.794424 | 0.677 | 10 | 5 | 0 | 5 | 0.572979 | 0.026143 | 0.582 | 0.991 | 1.000 |
| Cloud Admin Medium 3-gram Seq-only | 29.444148 | 0.781443 | 0.707 | 9 | 5 | 0 | 4 | 0.658056 | 0.031687 | 0.667 | 0.992 | 1.000 |
| Cloud Admin Medium 1-gram Seq-feats | 73.175335 | 0.815612 | 0.323 | 14 | 4 | 0 | 10 | -0.001401 | 0.005482 | 0.048 | 0.488 | 0.518 |
| Cloud Admin Medium 2-gram Seq-feats | 113.543062 | 0.872850 | 0.439 | 15 | 7 | 0 | 8 | -0.000645 | 0.007493 | 0.023 | 0.576 | 0.421 |
| Cloud Admin Medium 3-gram Seq-feats | 108.910944 | 0.866384 | 0.350 | 17 | 7 | 0 | 10 | 0.015151 | 0.014741 | 0.050 | 0.698 | 0.301 |

Table A.1 – Experimental results for sequence features as numerical data

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

**Titre :** Sécurité dans le cloud : Framework de détection de menaces internes basé sur l'analyse d'anomalies

**Mots clés :** Securité au Cloud,

**Résumé :** Le Cloud Computing (CC) ouvre de nouvelles possibilités pour des services plus flexibles et efficaces pour les clients de services en nuage (CSC). Cependant, la migration vers le cloud suscite aussi une série de problèmes, notamment le fait que, ce qui autrefois était un domaine privé pour les CSC, est désormais géré par un tiers, et donc soumis à ses politiques de sécurité. Par conséquent, la disponibilité, la confidentialité et l'intégrité des CSC doivent être assurées. Malgré l'existence de mécanismes de protection, tels que le cryptage, la surveillance de ces propriétés devient nécessaire. De plus, de nouvelles menaces apparaissent chaque jour, ce qui exige de nouvelles techniques de détection plus efficaces.

Les travaux présentés dans ce document vont au-delà du simple l'état de l'art, en traitant la menace interne malveillante, une des menaces les moins étudiées du CC. Ceci s'explique principalement par les obstacles organisationnels et juridiques de l'industrie, et donc au manque de jeux de données appropriés pour la détecter. Nous abordons cette question en présentant deux contributions principales.

Premièrement, nous proposons la dérivation d'une méthodologie extensible pour modéliser le comportement d'un utilisateur dans une entreprise. Cette abstraction d'un employé inclut des facteurs intra-psychologiques ainsi que des informations contextuelles, et s'inspire d'une approche basée sur les rôles. Les comportements suivent une procédure probabiliste, où les motivations malveillantes devraient se produire selon une probabilité donnée dans la durée.

La contribution principale de ce travail consiste à concevoir et à mettre en œuvre un cadre de détection basé sur les anomalies pour la menace susmentionnée. Cette implémentation s'enrichit en comparant deux points différents de capture de données : une vue basée sur le profil du réseau local de la entreprise, et une point de vue du cloud qui analyse les données des services avec lesquels les clients interagissent. Cela permet au processus d'apprentissage des anomalies de bénéficier de deux perspectives : (1) l'étude du trafic réel et du trafic simulé en ce qui concerne l'interaction du service de cloud computing, de manière de caracteriser les anomalies ; et (2) l'analyse du service cloud afin d'ajouter des statistiques prenant en compte la caractérisation globale du comportement.

La conception de ce cadre a permis de détecter de manière empirique un ensemble plus large d'anomalies de l'interaction d'une compagnie donnée avec le cloud. Cela est possible en raison de la nature reproductible et extensible du modèle. En outre, le modèle de détection proposé profite d'une technique d'apprentissage automatique en mode cluster, en suivant un algorithme adaptatif non supervisé capable de caractériser les comportements en évolution des utilisateurs envers les actifs du cloud. La solution s'attaque efficacement à la détection des anomalies en affichant des niveaux élevés de performances de clustering, tout en conservant un FPR (Low Positive Rate) faible, garantissant ainsi les performances de détection pour les scénarios de menace lorsque celle-ci provient de la compagnie elle-même.

**Title:** Security in the Cloud: An Anomaly-based Detection for Insider Threats

**Keywords:** Cloud Computing, Security, Intrusion Detection Systems, Anomaly Detection, Insider Threat

**Abstract:** Cloud Computing (CC) opens new possibilities for more flexible and efficient services for Cloud Service Clients (CSCs). However, one of the main issues while migrating to the cloud is that what once was a private domain for CSCs, now is handled by a third-party, hence subject to their security policies. Therefore, CSCs' confidentiality, integrity, and availability (CIA) should be ensured. Although the existence of protection mechanisms, such as encryption, the monitoring of the CIA properties becomes necessary. Additionally, new threats emerge every day, requiring more efficient detection techniques.

The work presented in this document goes beyond the state of the art by treating the malicious insider threat, one of the least studied threats in CC. The reason is mainly due to the organizational and legal barriers from the industry, and therefore the lack of appropriate datasets for detecting it. We tackle this matter by addressing two challenges.

First, the derivation of an extensible methodology for modeling the behavior of a user in a company. This abstraction of an employee includes intra psychological factors, contextual information and is based on a role-based approach. The behaviors follow a probabilistic procedure, where the malevolent motivations are considered to occur with a given probability in time.

The main contribution is the design and implementation of an anomaly-based detection framework for the threat mentioned above. This implementation enriches itself by comparing two different observation points: a profile-based view from the local network of the company, and a cloud-end view that analyses data from the services with whom the clients interact. This allows the learning process of anomalies to benefit from two perspectives: (1) the study of both real and simulated traffic with respect to the cloud service's interaction, in favor of the characterization of anomalies; and (2) the analysis of the cloud service in order to aggregate statistical data that support the overall behavior characterization.

The design of this framework empirically shows to detect a broader set of anomalies of the company's interaction with the cloud. The replicable and extensible nature of the mentioned insider model makes this possible. Also, the proposed detection model takes advantage of the autonomous nature of a clustering machine learning technique, following an unsupervised, adaptive algorithm capable of characterizing the evolving behaviors of the users towards cloud assets. The solution efficiently tackles the detection of anomalies by showing high levels of clustering performance, while keeping a low False Positive Rate (FPR), ensuring the detection performance for threat scenarios where the threat comes from inside the enterprise.