Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Máster Universitario en Inteligencia Artificial

Master's Final Project

Anomaly-based Network Intrusion Detection System Using Semi-supervised Models

Author: Paula Cordero Encinar
Supervisors: Concha Bielza Lozoya and Pedro Larrañaga Múgica

Madrid, July 2022

Master's Final Project
Máster Universitario en Inteligencia Artificial

Title: Anomaly-based Network Intrusion Detection System Using Semi-supervised Models

July 2022

Author:	Paula Cordero Encinar
Supervisors:	Concha Bielza Lozoya and Pedro Larrañaga Múgica
	Departmento de Inteligenci Artificial
	ETSI Informáticos
	Universidad Politécnica de Madrid

Acknowledgements

To my supervisors Concha Bielza and Pedro Larrañaga for their wise advise, confident and motivation that have guided me along the path of research.

To Dani Herreras and Borja Lanseros of *Titanium Industrial Security S.L.* for providing me the resources and assistance for this project, as well to introduce me in the amazing world of cybersecurity. Also to Carlos and Jorge for being my fellows along the way and for all the interesting discussions with them.

To my family and friends for their constant support.

This project has been developed in collaboration with *Titanium Industrial Security S.L.*, under the project entitled "Network SLIcing SEcurity for next generation communications (SLISE)", running 2021-2024, and funded by Centro para el Desarrollo Tecnológico Industrial (CDTI) within the Call "Misiones Ciencia e Innovación 2021".

Resumen

El crecimiento masivo de la cantidad de datos transmitidos entre múltiples dispositivos, utilizando diversos protocolos de comunicación ha planteado serios problemas de seguridad. Esto ha provocado un aumento en la importancia de desarrollar sistemas avanzados de detección de intrusos. En particular, en los entornos industriales donde la llamada Industria 4.0 ha aumentado la conectividad entre dispositivos, la seguridad de la red es de suma importancia, pero también especialmente desafiante debido a la gran cantidad de máquinas conectadas que es necesario proteger.

En este trabajo de fin de máster, presentamos un sistema de detección de intrusiones en red basado en anomalías utilizando técnicas semisupervisadas. Hemos aplicado dos modelos diferentes dentro del sistema y hemos comparado sus resultados y explicaciones. La idea general de nuestro sistema es modelar el comportamiento normal para posteriormente detectar los eventos maliciosos que difieren de dicho comportamiento. También hemos prestado especial atención a la interpretabilidad, ya que queremos que nuestro sistema de detección de intrusos no sólo sea eficiente, sino también útil para los analistas de ciberseguridad que tengan que examinar los eventos anómalos.

Abstract

The massive growth of data that are transmitted through a variety of devices and communication protocols have raised serious security concerns. This has increased the importance of developing advanced intrusion detection systems. In particular, in industrial environments where the so-called Industry 4.0 has increased the connectivity among devices, network security is of utmost importance, but also particularly challenging due to the large number of connected machines that must be protected.

In this master's thesis, we present an anomaly-based network intrusion detection system using semi-supervised techniques. We have applied two different models within the system and compared their performance and explanations. The general idea of our system is to model normal behaviour to subsequently detect malicious events that differ from normal behaviour. We have also paid special attention to interpretability, as we want our intrusion detection system not only to be efficient, but also helpful for cybersecurity analysts who may have to examine suspicious events.

Contents

Chapter 1

Introduction

1.1 Motivation

Nowadays, industrial control systems are experiencing a new revolution with the interconnection of the operational equipment thanks to the Internet, and the introduction of cutting-edge technologies such as cloud computing or big data within the organization. These and other technologies are paving the way to the Industry 4.0.

However, the advent of these technologies, and the innovative services that are enabled by them, have also brought novel threats. Malicious attacks have become more sophisticated and the foremost challenge is to identify unknown and obfuscated malware, as the malware authors use different evasion techniques for information concealing to prevent detection. We could highlight a set of stealthy and sophisticated hacking processes called advanced persistent threat (APT) [Chen *et al.*, 2018]. The APT intrusion kill chain security model has become popular to describe the stages of attacks. The APT intrusion kill chain relies on the premise that an attack has an operational life cycle for gathering information and exploiting the victim system. The steps in the chain relate to recent anomalous events covering a set of common actions in a targeted attack.

Therefore, it is crucial to take these novel threats into consideration and further study how to apply the necessary requirements in the design of effective and reliable intrusion detection mechanisms for the industry of the future. The aim is that it does not become a profitable target for attackers.

Even though, in recent years, increased attention has been paid to network security, which is evidenced by the growing body of literature available on this topic, there are still a lot of open research lines.

Among the tools used for network security are firewalls, antivirus software, and intrusion detection systems (IDSs), all of which seek to ensure the security of the network and all its associated assets within a cyberspace. Specifically, the network-based IDS is the attack detection mechanism that provides the desired security by constantly monitoring the network traffic for malicious and suspicious behaviour.

The idea of IDS was first proposed by Jim Anderson in 1980 [Anderson, 1980]. Since then, many IDS products were developed and matured to satisfy the needs of network security. However, as already mentioned, the immense evolution in the technologies over the last decade has resulted in a large expansion in the network size, and the number of applications handled by the network nodes. As a result, a huge amount of important data is being generated and shared across different network nodes. The security of these data and network nodes has become a challenging task due to the generation of a large number of new attacks. Therefore, almost every node within a network is vulnerable to security threats.

For instance, if the data of a node is very important for an industrial organization, any compromise to the node's information may cause a huge impact on that organization in terms of its market reputation and financial losses. Furthermore, in an industrial environment some network nodes are cyber-physical machines, so threats can also lead to serious physical consequences and major accidents, such as the malicious computer worm *Stuxnet* that ruined almost one-fifth of Iran's nuclear centrifuges in 2013. Existing IDSs have shown inefficiency in detecting various attacks including zero-day attacks, that is unknown attack on a vulnerability, and reducing the false alarm rates. This eventually results in a demand for an efficient, accurate, and cost-effective network IDS to provide high security to the network.

To fulfill the requirements of an effective IDS, researchers have explored the possibility of using machine learning (ML) and its branch of deep learning (DL) techniques. Both ML and DL come under the big umbrella of artificial intelligence and aim at learning useful information from the data. These techniques have gained enormous popularity in the field of network security over the last decade due to graphic processing units (GPUs). Both ML and DL are powerful tools in learning useful features from the network traffic and identifying abnormal activity.

1.2 Problem statement

Titanium Industrial Security S.L. monitors network traffic at client companies and aims to detect cyberattacks in a timely manner so that they can be dealt without causing further damage. It is important to remark, that their focus is on industrial companies, where these problems have not been as addressed as much as in IT environments and, as mentioned above, could lead to serious physical damage. They plan to deploy local IDSs throughout the client network, which will communicate with each other. The reason for using various local systems instead of a single centralized one is the alleviation of computational load and the possibility for these local devices to be faster and more efficient.

They proposed us to design and develop those local IDSs. Besides, they wanted the system to be able to provide further explanations of detection decisions, in case they are needed by a cybersecurity analyst.

From a general point of view, explainability is currently becoming an important issue for the widespread adoption of artificial intelligence systems [Leslie, 2019]. Some of these systems have been forbidden for certain applications due to their lack of transparency. That is why *Titanium* wants to emphasize the aspect of explainability.

1.3 Objectives

The general objective of this work is to develop an IDS to identify malicious events in network traffic communication in an industrial environment.

We will expand on our approach to achieve this goal in the body of this work, in short we have decided to apply semi-supervised anomaly detection methods. That is, we train the model with data on the normal behaviour of the network, and then, search for anomalous instances that deviate from that behaviour.

Futhermore, we have decided to work within the field of ML. The particular models we have used are Bayesian networks, specifically the semiparametric version, as well as, variational autoencoders, which are in the subfield of DL.

As we have already pointed out, explanations are a key aspect of the desired system for interpreting and analysing the results obtained. That is why, we have also devoted a great deal of effort to this task, trying to find the best way to do it. Our aim is not only to provide individual explanations for each anomalous event, but also to group similar suspicious events which belong to the same cyberattack. This is done for the shake of simplicity and unification of alarm alerts.

The particular objectives of this work can be listed as follows:

- To review the techniques that have been employed for anomaly-based intrusion detection, particularly in the field of cybersecurity.
- To determine which models are most useful for the given problem based on different factors, such as computational efficiency, performance and interpretability of the results.
- To propose a comprehensive network IDS, that can accurately identify malicious events in the network environment and provide an explanation for them. In addition, the system must be able to adapt to multiple network environments, as each client company has a particular network traffic.
- To reduce the number of false alarms in the system.
- To evaluate the capabilities of the proposed method in different datasets.
- To discuss possible deficiencies and suggest further improvements.

1.4 Document structure

In order to meet the objectives proposed in the previous section, this master's thesis is structured into five chapters:

- In this chapter, we have presented the context of the problem and its motivation. We have stated the exact problem we are addressing and outlined the main objectives of this thesis.
- Chapter 2 is devoted to the theoretical framework of our system and a review of the available literature in the field. We start with the discussion of basic concepts of IDSs and their types, centering our attention on anomaly-based detection which is the focus of our work. We then introduce the specifics of our two approaches to solve the problem: semiparametric Bayesian networks, and

their structure learning algorithms, and variational autoencoders. Finally, we provide an overview of explainability in the field of artificial intelligence, as we are not only concerned with developing an efficient and accurate system, but also with helping the cybersecurity analyst with valuable explanations of our system's decisions.

- Chapter 3 presents our proposed solution and shows how it operates. We provide a detailed explanation for each of the components of the system and justify certain design decisions.
- Chapter 4 shows our results when using our IDS on the attack dataset provided by *Titanium Industrial Security S.L.* along with the corresponding discussions and explanations of the type of attack. Besides, we also present the Bayesian network structures obtained when applying our algorithm to *Titanium*'s data.
- Finally, chapter 5 concludes our work with some final remarks and discusses relevant aspects of the system. In addition, we present proposals for future work that could enhance or improve our network anomaly-based IDS.

Chapter 2

Theoretical background

2.1 Cybersecurity: intrusion detection systems

An IDS is a system that monitors network traffic for suspicious activity and issues alerts when such activity is discovered. It is a software application that scans a network or a system for the harmful activity or policy breaching. Any malicious venture or violation is normally reported either to an administrator or collected centrally.

Although IDSs monitor networks for potentially malicious activity, they are also disposed to false alarms. Hence, organizations need to fine-tune their IDS products when they first install them. It means properly setting up the IDSs to recognize how normal traffic on the network looks like as compared to malicious activity. Meng and Kwok (2012) present an adaptative false alarm filter using ML for intrusion detection. The proposed architecture has the ability to adapt itself by selecting the best single-performance ML algorithm according to the specific contexts. In the evaluation, their adaptive false alarm filter keeps up a stable reduction rate of false alarms and filter out more than 80% of them.

IDSs can be classified by where detection takes place: network or host, that determines the type of data, or by the detection method that is employed: signature or anomaly-based.

In terms of the location where the activity is analysed, there are two common types:

- **Network intrusion detection system (NIDS)**. They are set up at a planned point within the network to examine traffic from all devices on the network.
- **Host-based intrusion detection system (HIDS)**. They run on independent hosts or devices of the network. A HIDS monitors the incoming and outgoing packets from the device.

Regarding the detection approach, we distinguish:

- **Signature-based detection**. A signature-based IDS identifies the attacks on the basis of the specific patterns which are already known. This method has the advantage of being fast and accurate.
- **Anomaly-based detection**. We devote next section to explain this approach, which is also used in other sectors besides cybersecurity. In contrast to signature-

based methods, it can achieve detection of unknown attacks. However, they are prone to detecting false positives.

2.2 Anomaly detection

Anomaly detection is a task which consists of finding events in the data that differ from the normal behaviour of the system under study. Despite that in the earliest days, detecting anomalies was motivated by data cleansing, soon more attention has been drawn toward outliers themselves as they often represent interesting and critical situations, such as, cyberattacks in a network or mechanical faults caused by defective industrial equipment, among others. Hence, plenty of research efforts have been devoted to developing high-performance anomaly detection techniques, which have been applied to a variety of real-life scenarios, including IDSs [García-Teodoro *et al.*, 2009; Iliyasu and Deng, 2022; Viegas *et al.*, 2017; Yeung and Chow, 2002], which is the topic of this work; fraud detection [Paula *et al.*, 2016; Thiprungsri and Vasarhelyi, 2011; Vanhoeyveld *et al.*, 2020]; medical anomaly diagnosis [Fernando *et al.*, 2021; Sedik *et al.*, 2019; Wong *et al.*, 2003]; fault detection in industrial environment [García, 2019; Martí *et al.*, 2015; Ogbechie *et al.*, 2017; Vercruyssen *et al.*, 2018]; and anomaly detection in urban traffic flow [Djenouri *et al.*, 2019].

There are many different surveys on this topic, which provide a complete overview of the subject [Boukerche *et al.*, 2020; Chandola *et al.*, 2009; Smiti, 2020], and in particular in the domain of IDSs [Liu and Lang, 2019; Khraisat *et al.*, 2019; Yang *et al.*, 2022]. Moreover, in the last years there has been a rise in the number of works using DL techniques for anomaly detection [Aldweesh *et al.*, 2020; Pang *et al.*, 2021], as well as works concerning anomaly detection in data streams [Boukerche *et al.*, 2020] or distributed approaches in the face of big data challenge to deal with the curse of dimensionality [Jain and Kaur, 2021].

According to Chandola *et al.* (2009), anomalies can be classified into three types:

- **Point anomalies**. If an individual data instance can be considered as anomalous with respect to the rest of the data, then the instance is termed a point anomaly. This is the simplest type of anomaly and is the focus of the majority of research on anomaly detection.
- **Contextual anomalies**. If a data instance is anomalous in a specific context, but not otherwise, then it is called a contextual anomaly (it can also be found in the literature as conditional anomaly).
- **Collective anomalies**. If a collection of related data instances is anomalous with respect to the entire dataset, it is termed a collective anomaly. The individual data instances in a collective anomaly may not be anomalies by themselves, but their occurrence together as a collection is anomalous.

The labels associated with a data instance denote whether that instance is normal or anomalous. It should be noted that obtaining labelled data that is accurate as well as representative of all types of behaviours, is often prohibitively expensive. Labeling is often done manually by a human expert and hence, substantial effort is required to obtain the labelled training dataset. Typically, getting a labelled set of anomalous data instances that covers all possible type of anomalous behavior is more difficult than getting labels for normal behavior. Moreover, the anomalous behaviour is often dynamic in nature, for example, new types of anomalies might arise, for which there is no labelled training data.

Based on the extent to which the labels are available, anomaly detection techniques can operate in one of the following three modes:

- **Supervised anomaly detection**. Techniques trained in supervised mode assume the availability of a training dataset that has labelled instances for normal as well as anomaly classes. A typical approach in such cases is to build a predictive model for normal versus anomaly classes. Any unseen data instance is compared against the model to determine which class it belongs to. There are two major issues that arise in supervised anomaly detection. First, the anomalous instances are far fewer compared to the normal instances in the training data. Issues that arise due to imbalanced class distributions have been addressed in literature [Chawla, 2009; Kamalov, 2020; Provost, 2000]. Second, obtaining accurate and representative labels, especially for the anomaly class is usually challenging. A number of techniques have been proposed that inject artificial anomalies into a normal dataset to obtain a labelled training dataset [Fan *et al.*, 2004; Lomio *et al.*, 2020].
- **Semi-supervised anomaly detection**. Methods that operate in a semi-supervised mode [Villa-Pérez *et al.*, 2021] typically assume that the training data has labelled instances only for the normal class. This is properly called one-class classification. Since they do not require labels for the anomaly class, they are more widely applicable than supervised techniques.
- **Unsupervised anomaly detection**. Techniques that operate in unsupervised way do not require training data, and thus, are the most widely applicable [Falcão *et al.*, 2019; Zoppi *et al.*, 2021]. The techniques in this category make the implicit assumption that normal instances are far more frequent than anomalies in the test data. If this assumption is not true, then such techniques suffer from high false alarm rate. Many semi-supervised techniques can be adapted to operate in an unsupervised mode by using a sample of the unlabelled dataset as training data. Such adaptation assumes that the test data contains very few anomalies and the model learned during training is robust to these few anomalies.

An important aspect for any anomaly detection technique is the manner in which the anomalies are reported. Generally, the outputs produced by anomaly detection techniques could be a label, indicating whether the instance is normal or anomalous, or a score, that quantifies the degree to which an instance is considered an anomaly. Thus, the output of such techniques is a ranked list of anomalies. An analyst may choose to either analyze the top few anomalies or use a cutoff threshold to select the anomalies. Ghafouri *et al.* (2016) study the problem of finding optimal detection thresholds for anomaly-based detectors implemented in dynamical systems in the face of strategic attacks, using a game-theoretic setup.

Finally, we provide a categorization of anomaly detection models and review for each type the existing literature for network intrusion detection applications, which is the focus of this work.

- **Classification-based anomaly detection**

  A classifier is learnt from labelled data to discriminate between normal and anomalous data. These models usually require supervised data. Although, labeling is time-consuming and tedious for large datasets, there are many publicly available datasets which are already labelled. In the field of cybersecurity we can highlight KDD 99 [Tavallaee *et al.*, 2009], UNSW-NB15 [Moustafa and Slay, 2015], NSL-KDD [Dhanabal and Shantharajah, 2015], LANL dataset [Turcotte *et al.*, 2018] and CSE-CIC-IDS-2018 [Leevy and Khoshgoftaar, 2020].

  Among the different supervised anomaly detection techniques for network intrusion detection, we have support vector machines (SVMs) that have high generalization capability and perform well with high-dimensional data used in intrusion detection [Jan *et al.*, 2019; Sakr *et al.*, 2019]; decision trees that are also a common model thanks to their intuitive classification strategy, interpretability and simplicity to implement, however they have the disadvantage of weak robustness [Abbes *et al.*, 2010; Malik and Khan, 2018]; and Bayesian network classifiers, such as the basic naïve Bayes (NB), where, despite the violation of its conditional independence assumption, relative high accuracy is achieved [Koc *et al.*, 2012; Mehmood *et al.*, 2018] and the average one-dependence estimator (AODE) which solves the attribute independency problem in NB [Jabbar *et al.*, 2017].

  Besides, we have also ensemble methods which improve the generalizability and accuracy of the final model by combining multiple classifiers, and are less likely to be overfitted. Some examples are shown in Ennaji *et al.* (2021), Gao *et al.* (2019), Jabbar *et al.* (2017) and Teng *et al.* (2018).

  In addition, we have nearest neighbour-based anomaly detection methods which can be seen as a subgroup of classification-based models. The underlying assumption of these techniques is that normal data instances are closer to their neighbours, thus forming a dense neighbourhood, whereas outliers are far from their neighbours, thus sparsely populated. Distance or similarity between two data instances can be computed in different ways.

  Ertoz *et al.* (2004) present MINDS anomaly detection module, that assigns the degree of being an outlier to each data point, which is called the local outlier factor [Breunig *et al.*, 2000]. Their approach can detect outsider and insider attacks and situations where a virus or worm has entered the network environment.

  Nearest neighbour-based methods have the advantage of a more refined granularity on the outlier analysis over clustering-based approaches (see below). This enables nearest neighbour-based methods to differentiate between strong outliers and weak outliers that are more likely to be considered as noise [Aggarwal, 2017]. Nevertheless, due to their high computational complexity, not many recent works use this approach.

- **Clustering-based anomaly detection**

  Algorithms based on clustering usually take a two-step procedure: first, grouping the data with clustering algorithms and then, analyzing the degree of deviation based on the clustering results. Compared with nearest neighbour-based approaches, a major advantage of clustering-based outlier detection is its efficiency.

Different clustering methods have been applied in the field of NIDS. Tian and Jianwen (2009) propose an approach based on a modified version of k-means clustering, which leads to an improvement in accuracy with respecto to the classical algorithm. Aiming at the problem that traditional network intrusion detection algorithms have of low detection efficiency and high false alarm rate, Xiaofeng and Xiaohong (2017) present an algorithm based on improved k-means and multi-level SVM. Li *et al.* (2018) apply density peaks clustering for network instrusion detection, which does not need many parameters and whose iterative process is based on point density. Ariyaluran Habeeb *et al.* (2019) develop streaming sliding window local outlier factor coreset clustering algorithms to perform real-time detection. Their method proves higher accuracy, and lesser memory consumption and execution time compared to existing methods.

- **Statistical anomaly detection**

  The underlying principle of these models is that normal data instances occur in high probability regions of a stochastic model, while anomalies occur in the low probability regions of the stochastic model. Statistical techniques fit a statistical model, usually for normal behaviour, to the given data and then apply a statistical inference test to determine if an unseen instance belongs to this model or not. Instances that have a low probability of being generated from the learned model, based on the applied statistical test, are declared as anomalies. Both parametric as well as nonparametric techniques have been applied to fit a statistical model. While parametric techniques assume the knowledge of the underlying distribution and estimate the parameters from the given data, nonparametric techniques do not generally assume knowledge of the underlying distribution. Semiparametric techniques have also been developed taking the best of both worlds.

  In terms of parametric models, assuming that the data is Gaussian distributed is a simple choice, however it is not always true. The methodologies most commonly used are probabilistic graphical models, such as Kalman or particle filters [Ayyarao and Kiran, 2021], hidden Markov models [Jing *et al.*, 2021] or Bayesian networks (BNs); finite mixture models; and regression-based models [Evangelou and Adams, 2020]. In BNs, anomalies are detected because a low probability is assigned to them by the network. Depending on the type of data a particular type of BN is selected, such as static BNs [Sun *et al.*, 2018], dynamic BNs (DBNs) [An *et al.*, 2006] or continuous time BNs (CTBNs) [Xu and Shelton, 2010], among others. On the other hand, finite mixture models can be defined as a convex combination of two or more probability distributions functions (PDFs), the joint properties of which can approximate any arbitrary distribution. They are a powerful and flexible probabilistic modelling tool for univariate and multivariate data in the cybersecurity scenario [Greggio, 2013; Moustafa *et al.*, 2017].

  Regarding non-parametric methods, the most common models are histograms and kernel density estimations (KDEs) [Parzen, 1962; Rosenblatt, 1956]. Despite their simplicity, histogram-based models have been popular in the intrusion detection community, since the behaviour of the data is goberned by certain profiles (user or software) that can be efficiently captured [Eskin, 2000; Goldstein and Dengel, 2012; Kind *et al.*, 2009]. The KDE method bases its estimations on some kernel distributions, such as Gaussian, for all the sample space data and then integrates the local contributions of all the distributions. For example, Shen and Agrawal (2006) suggest a NIDS based on a non-parametric method which simulates the PDFs of some random variables. To classify malicious and normal instances, a set of KDEs is established and the distribution parameters estimated. This method is extended by Caudle *et al.* (2015) to build a non-stationary high-dimensional PDF estimator using parallel programming to identify computer intrusions. Recently, Hu *et al.* (2020) proposed a new kernel function to estimate samples' local densities and a weighted neighborhood density estimation to increase the robustness to changes in the neighborhood size, outperforming Gaussian kernel and the Epanechnikov kernel functions on KDD 99 dataset.

- **Information theoretic anomaly detection**

  These techniques assume that anomalies in data induce changes in the information content of the dataset, which is composed of a variety of information theory measures such as entropy, relative entropy [Noble and Cook, 2003] or Kolmogorov complexity [Kulkarni and Bush, 2006].

- **Deep learning anomaly detection**

  DL comprises a class of techniques based on deep artificial neural networks. Recently, DL methods are widely used due to their capacity to produce highly nonlinear models which capture complex relationships in data.

  Different architectures can be employed in anomaly-based detection approaches based on DL, including generative, discriminative, and hybrid methods. The generative architecture computes joint probability distributions from observed data with their classes, which involves the following models: autoencoder (AE) [Hannan *et al.*, 2021; Shone *et al.*, 2018], generative adversarial network (GAN) [Iliyasu and Deng, 2022], restrictive Boltzmann machine (RBM) [Fiore *et al.*, 2013], deep belief network (DBN) [Peng *et al.*, 2019; Wang *et al.*, 2021], and recurrent neural network (RNN) and their variants like long short-term memory (LSTM) [Imrana *et al.*, 2021] and gated recurrent unit (GRU) [Agarap, 2018; Singh *et al.*, 2021]. On the other hand, the discriminative architecture estimates posterior distributions of classes conditioned on the observed data, that comprises classical deep neural networks (DNNs) [Tang *et al.*, 2016] and convolutional neural networks (CNNs) [Wei *et al.*, 2017; Xiao *et al.*, 2019]. Ensemble methods that combine different architectures have also been proposed [Ludwig, 2017, 2019].

2.3 Semiparametric Bayesian networks

Bayesian networks [Koller and Friedman, 2009; Pearl, 1988] are probabilistic graphical models that represent a set of random variables and their conditional dependencies. They take advantage of the conditional independences present in the probability distributions to model them in a factorized form. Bayesian networks consist of a qualitative component, represented by a directed acyclic graph (DAG) structure and a quantitative component, represented by tables of conditional probabilities, as illustrated in Figure 2.1. They are a valuable tool for computer technology and artificial intelligence that allow to model uncertainty by dealing with probability distributions.

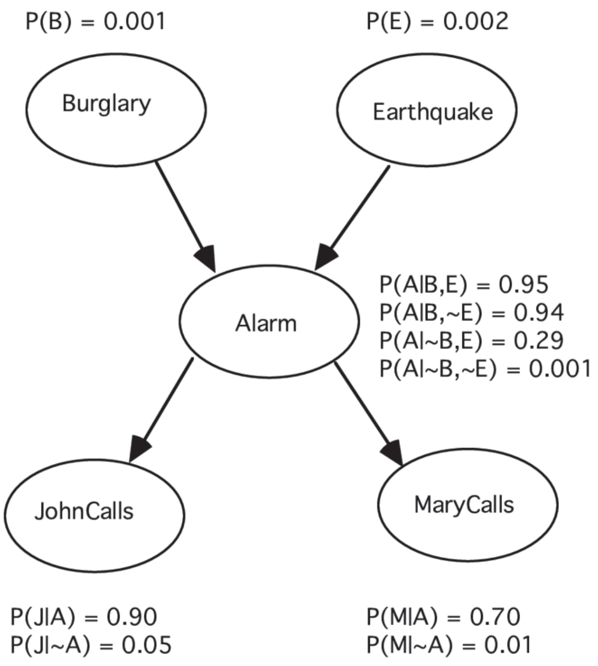


Figure 2.1: Example of Bayesian network [Pearl, 1988].

First of all, we introduce some notation and terminology regarding Bayesian networks, that will be helpful for this section. We denote with capital letters, e.g. X, a random variable, while using the boldface version to represent random variables vectors. A dataset is represented by $\mathcal{D} = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$, that contains N different instances. In terms of the graphical representation, $\mathcal{G} = (V, A)$ defines a DAG with a set of nodes V and a set of arcs $A \subseteq V \times V$. The set of nodes V can index a vector of random variables $\mathbf{X}$. The set of parents of a node i is denoted $\text{Pa}(i)$, so the respective parent random variables are $\mathbf{X}_{\text{Pa}(i)}$.

The semiparametric version of Bayesian networks handles continuous variables and combines parametric and non-parametric conditional probability distributions (CPDs), thus being more flexible and robust. The aim of semiparametric Bayesian networks (SPBNs) is to incorporate the benefits of both components: on the one hand, the bounded complexity of parametric models and, on the other hand, the flexibility of non-parametric ones. Hence, they provide a further generalization of Gaussian Bayesian networks (GBNs) [Geiger and Heckerman, 1994; Shachter and Kenley, 1989], where variables are continuous and all of the CPDs are linear Gaussians, and KDE Bayesian networks (KDEBNs) [Hofmann and Tresp, 1995], that estimate the true density of the continuous variables using kernels. Furthermore, the hybrid version of SPBNs allows to combine continuous and discrete variables thus offering a great number of advantages over other models.

Not much work has been done in this field. Boukabour and Masmoudi (2021) define a continuous SPBN where all of the CPDs are semiparametric regressions. The structure learning algorithm they introduce is based on statistical hypothesis tests of conditional independence that has the drawback of requiring the knowledge of the correct ancestral ordering for the nodes. Atienza (2021) and Atienza *et al.* (2022c) present a generalization of SPBNs and introduces a hybrid version (HSPBN) in Atienza *et al.*, 2022a. To the best of our knowledge, only they have explored the hybrid model of SPBNs.

We now explain in detail the representation and structure learning algorithms for SPBNs. Subsequently, we introduce HSPBNs that will be a fundamental model for this work, as we have to deal with continuous and categorical data.

2.3.1 Representation of SPBNs

SPBNs are composed of parametric and non-parametric CPDs. Whereas a parametric CPD can be used to represent linear relationships between random variables using a linear Gaussian distribution as in GBNs, non-parametric CPDs are considered to represent non-linear relationships given the flexibility of non-parametric models. The latter are defined as the ratio of two joint KDE models. Hereafter we will refer to this type of CPDs as conditional KDE distributions, which is defined for a random variable X_i, given $\mathbf{X}_{\text{Pa}(i)}$ as:

$$\hat{f}_{\text{CKDE}}(x_i|\mathbf{x}_{\text{Pa}(i)}) = \frac{\hat{f}_{\text{KDE}}(x_i, \mathbf{x}_{\text{Pa}(i)})}{\hat{f}_{\text{KDE}}(\mathbf{x}_{\text{Pa}(i)})} = \frac{\sum_{j=1}^{N} K_{\mathbf{H}}\left(\begin{bmatrix} x_i \\ \mathbf{x}_{\text{Pa}(i)} \end{bmatrix} - \begin{bmatrix} x_i^j \\ \mathbf{x}_{\text{Pa}(i)}^j \end{bmatrix}\right)}{\sum_{j=1}^{N} K_{\mathbf{H}_{-i}}\left(\mathbf{x}_{\text{Pa}(i)} - \mathbf{x}_{\text{Pa}(i)}^j\right)}, \tag{2.1}$$

where $\hat{f}_{\text{KDE}}(x_i, \mathbf{x}_{\text{Pa}(i)})$ and $\hat{f}_{\text{KDE}}(\mathbf{x}_{\text{Pa}(i)})$ are KDE models (further details can be found in Parzen (1962) and Rosenblatt (1956)), x_i and $\mathbf{x}_{\text{Pa}(i)}^j$ are the values of the j-th training instance for the variables X_i and $\mathbf{X}_{\text{Pa}(i)}$, respectively, and $\mathbf{H}$ and $\mathbf{H}_{-i}$ are the bandwidth matrices for the KDE models $\hat{f}_{\text{KDE}}(x_i, \mathbf{x}_{\text{Pa}(i)})$ and $\hat{f}_{\text{KDE}}(\mathbf{x}_{\text{Pa}(i)})$, respectively.

It is important to remark that unlike GBNs, linear Gaussian CPDs in SPBNs do not make assumptions about the normality of parent's random variables. Therefore, the unconditional distribution of random variables following the linear Gaussian conditional distribution X_i and $(X_i, \mathbf{X}_{Pa(i)})$ may not be necessarily multivariate normal distributions.

The graph of the SPBN model (Figure 2.2) contains the type of each node, which determines the type of corresponding CPD. There are no restrictions on the arcs, so the parent sets of each variable can be of different types: only linear Gaussian parents, only conditional KDE or a mixture of both.

2.3.2 Structure learning

The structure of a Bayesian network can be learnt automatically from data. There are three main approaches to learn the structure of a Bayesian network: score and search, constraint-based and hybrid procedures. The constraint-based methods are based on performing conditional independence tests and reconstructing a Bayesian network structure by representing the same tested conditional independences as accurately as possible. The score and search approaches rely on defining a scoring function that measure how well the Bayesian network structure fits to the training data. Atienza *et al.* (2022c) adapt the greedy hill-climbing (HC) algorithm, a score and search procedure, and also the PC algorithm, a constraint-based method for SPBN structure learning.

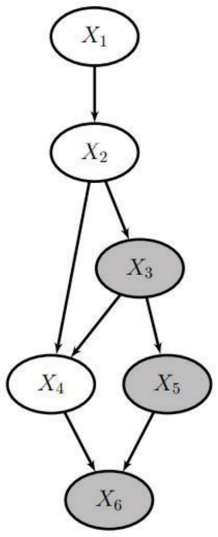


Figure 2.2: Structure of an example of SPBN [Atienza *et al.*, 2022c]. White nodes are of the linear Gaussian type, and gray shaded nodes are of the conditional KDE type.

2.3.2.1 Greedy hill-climbing

The HC algorithm moves through the space of possible structures applying operators that make local changes on a candidate structure to find a local optimal structure. At each iteration of the algorithm, the operator that produces the largest improvement in score is applied to generate a new candidate structure. The classical operators used in HC are arc addition, arc removal and arc reversal. However, in SPBNs, the structure is composed of arcs in a graph but also of the types of nodes. Then, a new operator called node type change is added into the HC algorithm for the learning of the semiparametric model.

Besides, another key element for score and search algorithms is the definition of a score function. For GBNs the typical ones are BIC [Schwarz, 1978], based, in part, on the likelihood function, and including a penalty term to avoid overfitting; and BGe, introduced by Geiger and Heckerman (2002) for learning GBNs, which is based on the posterior probability of the data given the structure and implies that the prior distribution over the parameters, θ, must be normal-Wishart. Their formulas are:

$$\mathcal{S}_{BIC}(\mathcal{D},\mathcal{G}) = \mathcal{L}(\mathcal{G},\hat{\theta}^{MLE} : \mathcal{D}) - \frac{\log N}{2}\text{Dim}(\mathcal{G}),$$
$$\mathcal{S}_{BGe}(\mathcal{D}|\mathcal{G}) = \int_{\Theta} P(\mathcal{D}|\mathcal{G},\theta)P(\theta|\mathcal{G})\, d\theta,$$

where $\mathcal{L}(\mathcal{G},\hat{\theta}^{MLE} : \mathcal{D})$ is the log-likelihood of the data when the parameters are estimated using maximum likelihood estimation with $\mathcal{D}$, $\text{Dim}(\mathcal{G})$ counts the number of free parameters in the structure $\mathcal{G}$, $P(\mathcal{D}|\mathcal{G},\theta)$ represents the likelihood of the data given the BN (structure and parameters) and $P(\theta|\mathcal{G})$ is the prior distribution over the parameters.

Nevertheless, for an SPBN, any score including the log-likelihood of the training data (like BIC) is inappropiate because the training data constitute part of the KDE model, as can be observed in Equation (2.1). Thus, the data used to fit the CPDs must be different from the data to evaluate the goodness of the model. Applying a k-fold cross validation to the data helps to attain this objective while assuring that all the data are used. So the HC algorithm will use a k-fold cross-validated likelihood score given by the following expression:

$$\mathcal{S}_{CV}^{k}(\mathcal{D}, \mathcal{G}) = \sum_{m=1}^{k} \mathcal{L}(\mathcal{G}, \theta^{\mathcal{I}^{-m}} : \mathcal{D}_{\downarrow \mathcal{I}^{m}}), \tag{2.2}$$

where $\mathcal{L}$ is the log-likelihood function, $\mathcal{I}^{m}$ denotes the instance indices for the m-th fold, $\mathcal{D}_{\downarrow \mathcal{I}^{m}}$ the data in that fold, $\mathcal{I}^{-m}$ comprises the indices not in the m-th fold and $\theta^{\mathcal{I}^{-m}}$ are the parameters estimated with the data $\mathcal{D}_{\downarrow \mathcal{I}^{-m}}$ that define the CPDs.

This score is valid for SPBNs because the log-likelihood is calculated over data that were not seen at the time of estimating the parameters. It can be understood as an estimator of the expected log-likelihood.

Furthermore, the score has the property of decomposability, that is, it can be expressed as the sum of local score terms related to each node and its parents given a selection of indices of disjoint folds of data. Hence, to take advantage of decomposability during the learning stage, we need to fix a specific set of indices $\mathcal{I}$. However, this can lead to overfitting that particular set of indices. To avoid this, the data is split into two disjoint sets, $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{validation}}$. The learning process is guided by $\mathcal{D}^{\text{train}}$ and a fixed set of indices over that dataset, whereas the subset $\mathcal{D}^{\text{validation}}$ controls the overfitting to that specific set of indices. Thus, the selection of new operators in HC is done using the score $\mathcal{S}_{CV}^{k}(\mathcal{D}^{\text{train}}, \mathcal{G})$, while overfitting is controlled measuring the goodness of the new structure at each iteration over $\mathcal{D}^{\text{validation}}$.

As previously mentioned, HC runs until reaching a local optimum where there is no operator which improves the current structure. However, the implementation of HC by Atienza *et al.* (2022c) relaxes this restriction, by allowing the structure not to improve the score for a maximum of λ iterations. The parameter λ is named patience. If the patience is a positive integer for trying to escape the local optimum, exploration beyond the local neighbourhood is allowed by implementing tabu search [Glover and Laguna, 1997], which forbids applying operators that reverse recently applied ones.

2.3.2.2 PC algorithm

The PC algorithm [Spirtes *et al.*, 2001] learns the structure of the Bayesian network performing conditional independence tests to construct the graph that best captures the conditional independence relationships.

The PC algorithm has two main steps. In the first step, it learns from data a skeleton graph, which contains only undirected edges. In the second step, it orients the v-structures $X_i \rightarrow X_k \leftarrow X_j$, whenever X_k does not belong to the set of variables that d-separates [Geiger *et al.*, 1990] X_i and X_j. The result of the PC algorithm is a partially DAG that represents the skeleton of an equivalence class. This graph is converted into a DAG of that equivalence class using the procedure proposed by e.g., Dor and Tarsi (1992).

One of the essential aspects in a constraint-based algorithm is the type of conditional independence test. A common option is the use of the PLC test, which assumes that all the variables follow a multivariate Gaussian distribution. Taking into account that SPBNs do not make any assumption on the distribution of any variable, non-parametric conditional independence tests need to be used. Atienza *et al.* (2022c) suggest using the randomized conditional correlation test [Strobl *et al.*, 2018], which is faster than other options as its distribution under the null hypothesis can be approximated with less computational resources.

Besides, to learn a SPBN, we need to determine the best type of CPD for each variable given the DAG learned by PC. An attractive approach would be to perform a statistical normality test, however due to computational burden, a more suitable option is to select the node types with the execution of the HC algorithm restricting the operators set to the node type change operator, hence, keeping the arc structure provided by the PC algorithm.

2.3.3 Hybrid semiparametric Bayesian networks

HSPBNs extend the support of SPBNs to model categorical data as well. Hence, continuous variables can also have discrete variables as parents, while allowing their CPDs to be designed using either parametric or non-parametric estimation models. An example of the structure of a HSPBN is shown in Figure 2.3.

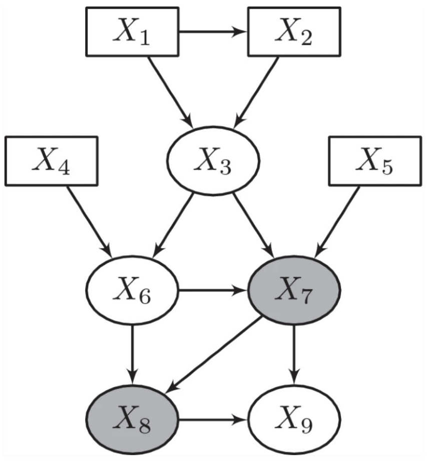


Figure 2.3: Example representing the structure of an HSPBN [Atienza *et al.*, 2022c]. Discrete variable nodes are represented with rectangles and continuous variable nodes with ellipses. Parametric CPD nodes are represented with white nodes and nonparametric CPDs nodes are represented with gray shaded nodes.

Thus, HSPBNs can model hybrid probability distributions which combine discrete and continuous random variables. Discrete variables are conditionally distributed as a categorical distribution, depicted by a conditional probability table (CPT). Likewise conditional linear GBNs, discrete variables cannot have continuous variables as parents in the graph. On the other hand, the CPD of continuous variables can be either parametric or non-parametric. Parametric ones assume that, for each possible configuration of the discrete parents, the continuous variables are conditionally distributed as a normal distribution and have a linear relationship with continuous parents. Then a parametric CPD in this type of networks is a conditional linear Gaussian CPD, which is composed of a linear Gaussian CPD for each discrete configuration of evidence. Non-parametric CPDs are based on the conditional KDE CPDs previously described, that do not make any assumptions about the marginal or conditional distribution of the variable. Nevertheless, conditional KDE only supports continuous evidence. The hybrid conditional KDE, similarly to conditional linear Gaussian CPDs, supports also discrete evidence by defining a conditional KDE for each discrete evidence configuration.

The structure learning of HSPBNs is based on the learning process presented in Section 2.3.2. In this case, the novel node type change operator previously introduced can transform conditional linear Gaussian CPDs into hybrid conditional KDE CPDs and vice versa.

The cross-validated score defined in Equation (2.2) can be applied directly for the hybrid version. The learning method that we will use for this type of network is the HC algorithm, but due to the restrictions of HSPBNs regarding the parents of discrete variables, the operators "add arc" or "arc reverse" that produce arcs from a continuous variables to a discrete one are forbidden.

2.4 Variational autoencoder

Given the large amount of data being constantly collected by the network routers, DL models have attracted a lot of attention in the cybersecurity field due to their ability to handle big datasets as well as to train realtime in a streaming manner, while retaining high performance. Additionally, DL models like the variational autoencoder (VAE) are shown to be robust to noisy data and adversarial attacks [Barrett *et al.*, 2022; Camuto *et al.*, 2021], and thus especially suitable for modeling network flows which are very noisy in nature. Although DL models are often criticized for their lack of explainability, recent advances have brought forward better understanding of these models. Some state-of-the-art techniques for explainability will be introduced in the following section.

In the following, we first describe the autoencoder (AE) model since the VAE has the same deep architecture as the AE. After, we present the specifics of VAE, a probabilistic generalization of the AE, that has been shown to be more flexible and robust [Jinwon and Sungzoon, 2015].

2.4.1 Autoencoder

An autoencoder has three main layers which correspond to the input layer to take in the features, the latent representation layer of the features, and the output layer which is the reconstruction of the features. The AE consists of two parts called encoder and decoder respectively, as illustrated in Figure 2.4. The encoder maps the input into its latent representation while the decoder attempts to reconstruct the features back from the latent representation. The encoder may be deep in the sense that information from the input is passed through several mappings and hidden layers, similarly to the deep architecture in a supervised DL model, and likewise for the decoder. It is important to note that as shown in Figure 2.4 the sizes of the encoder and decoder layers are generally chosen to be symmetrical.

The links between the layers show that the n values of the i-th layer $\mathbf{h}_i$ (that coincide with its number of nodes) can be computed as:

$$\mathbf{h}_i = g(\mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{b}_i), \tag{2.3}$$

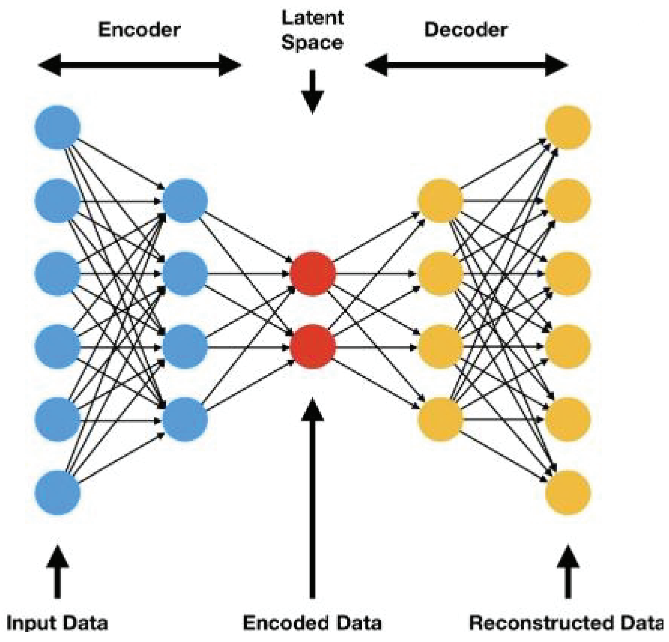


Figure 2.4: Autoencoder architecture [Lopez Pinaya *et al.*, 2020].

where $\mathbf{h}_{i-1}$ is a vector of m values for the previous layer, $\mathbf{W}_i$ is a matrix of weights that represents the relationship with the previous layer, whose size is $n \times m$, and $\mathbf{b}_i$ is a vector of bias terms, whose length is the same as that of the vector $\mathbf{h}_i$. Both $\mathbf{W}_i$ and $\mathbf{b}_i$ are parameters to be learned during the training stage. Finally, $g()$ is the activation function which is a non-linear transformation of its input, thus allowing complex relationships to be learned. The most common form for the activation function is the sigmoid function, $g(x) = (1 + e^{-x})^{-1}$, and the rectified linear unit (RELU), $g(x) = max(0, x)$. The learning of the parameters is generally achieved by minimizing the reconstruction errors (like, mean square errors) via backpropagation with random initialization, and can be optimized in different ways (optimization details can be found in Bengio (2012)).

Basically, AE can be understood as a deterministic model that maps a set of input features into their reconstruction. This is the aspect that differentiates AE from generative model variants of deep neural networks, such as the VAE and the GAN [Goodfellow *et al.*, 2014].

2.4.2 Variational autoencoder

Unlike AE which deterministically encodes the inputs into their latent representation and subsequently produce a reconstruction, the VAE [Kingma and Welling, 2014] is a generative model that treats the latent representation layer as random variables conditional on the inputs. Although the encoder and decoder in the VAE follow the same computational model as the AE, given by Equation (2.3), the encoding process is instead used to compute the parameters for the conditional distributions of the latent representation. Afterwards, the parameters can then be used to generate or sample the latent representation for decoding. The conditional distributions of continuous variable nodes are generally assumed to be Gaussian, but not necessarily.

The probabilistic nature of the VAE also means that we cannot simply employ the usual learning algorithm on standard objective function (such as mean square error) to train the model. Instead, a class of approximate statistical inference methods are used, which is called the variational Bayes (giving rise to the name VAE). In a simple way, an alternative objective function known as the variational lower bound is optimized, and stochastic sampling is used for approximation. For further details regarding inference methods we refer the reader to Kingma and Welling (2014). For the intermediate layers of the encoder and decoder, the RELU activation function is used, whereas the linear activation $g(x) = x$ is employed for the output, which provides the reconstructed vector.

2.5 Explainability

Artificial intelligence and in particular, the explainability thereof, has gained phenomenal attention over the last few years, due to the lack of ability of many data-driven artificial intelligence systems to provide information about the rationale behind their decisions to their users, which can be a major drawback, specially in critical domains such as those related to cybersecurity. The manifestation of explainable systems in high-risk areas has influenced the development of explainable artificial intelligence (XAI) in the sense of prescriptions or taxonomies of explanation. These include fairness, accountability, transparency and ethicality [Leslie, 2019]. The foundation of such a system should include these prescriptions such that a level of usable intelligence is reached to not only understand model behaviour but also understand the context of an application task [Barredo Arrieta *et al.*, 2020].

It is important to remark the difference between interpretability, which is the capacity of a model to be understandable to humans, and explainability. These two concepts are closely tied and many authors even do not differentiate between them [Carvalho *et al.*, 2019]. On the contrary, Rudin (2019) draws a clear line between interpretable and explainable ML: interpretable ML focuses on designing models that are inherently interpretable; whereas explainable ML tries to provide post hoc explanations for existing black box models, aiming to provide further information about the model by uncovering the importance of its parameters. Lipton (2018) stresses the difference in questions the two families of techniques try to address: interpretability raises the question “How does the model work?”; whereas explanation methods try to answer “What else can the model tell me?”.

Among inherent interpretable ML models, we focus in this work on Bayesian networks [Mihaljević *et al.*, 2021]. As already mentioned, Bayesian networks are probabilistic graphical models that can be used as a tool to manage uncertainty. Furthermore, Bayesian networks are capable of combining expert knowledge and statistical data, therefore allowing for complex scenarios to be modelled. These graphical models allow the user to reason about uncertainty in the problem domain by updating his/her beliefs, whether this reasoning occurs from cause to effect, or from effect to cause [Derks and de Waal, 2020]. Reasoning in Bayesian networks is often referred to as what-if questions. The flexibility of a Bayesian network allows for these questions to be predictive, diagnostic and inter-causal. Some what-if questions might be intuitive to formulate, but this is not always the case especially on a diagnostic and inter-causal level. This might result in sub-optimal use of explainability in Bayesian networks, especially on an end-user level. Apart from well-established reasoning methods, the probabilistic framework of a Bayesian network also allows for explainability in evidence. These include most probable explanation and most relevant explanation.

On the other hand, post-hoc methodologies can be further classified according to different factors:

- **Method**. There are two types of methods to provide explanations: local ones give explanations via individual instances or groups of nearby instances, and global ones describe the behaviour of models as a whole.
- **Model**. Approaches to provide explanations can be classified as model-agnostic, which apply to any model (based only on inputs and outputs), and model-specific, which are restricted to a specific model.

Recently, there has been a surge of research on enhancing the explainability in the field of cybersecurity, with a particular focus on explainability of DL models that have proven to be successful [Holder and Wang, 2021; Mahdavifar and Ghorbani, 2020; Szczepański *et al.*, 2020; Vigano and Magazzeni, 2018].

Chapter 3

Development

3.1 Data description

The synthetic data used in this project was simulated *ad hoc* by *Titanium Industrial Security S.L.* using virtual machines. Data consist of raw traffic information moving accross a computer network. We were provided with two different datasets:

- **normal dataset**, that comprises normal behaviour of the network.
- **attack dataset**, that captures two cyberattacks: a port scanning attack followed by a lateral movement attack on two different devices.

Port scanning is a method attackers use to scope out their target environment by sending packets to specific ports on a host and using the responses to find vulnerabilities and understand which services, and service versions, are running on a host. Thus, port scanning tries to classify ports into one of three categories: open, closed or filtered.

Lateral movement [Kang *et al.*, 2020] refers to the techniques that a cyberattacker uses, after gaining initial access, to move deeper into a network in search of sensitive data and other high-value assets. After entering the network, the attacker maintains ongoing access by moving through the compromised environment and obtaining increased privileges using various tools. Lateral movement is a sophisticated tactic that distinguishes today's advanced persistent threats from simplistic cyberattacks of the past. The first stage of lateral movement is reconnaissance, during which the attacker observes, explores and maps the network, its users, and devices. In our case, this phase is accomplished through a port scanning attack.

The data provided are in the form of raw text extracted directly from the *pcap* format. Each instance of the datasets contains the information of a sent packet. An example of a raw event is included in Appendix A. The total duration in seconds of the simulations and the number of recorded packets for both datasets are included in Table 3.1.

The main limitation of the data is the lack of labels for the attack dataset, although for the model development they are not necessary, as we use semi-supervised methods. They are of great help in validating the performance of the model. Therefore, we needed the help of a cybersecurity expert for the validation stage.

Dataset	Duration (seconds)	N° of events
normal dataset	7347.017	885508
attack dataset	6333.394	326941

Table 3.1: Details of the normal and attack datasets.

The simulated computer network environment is formed by eight different machines, (whose details are shown in Table 3.2) that not only communicate with each other, but also with external devices.

IP	Operative system	Description
10.0.2.4	Windows Server 2008	Office user
10.0.2.5	Windows Server 2008	Office user and victim
10.0.2.6	Windows Server 2008	Office user
10.0.2.7	Windows Server 2008	Office user and victim
10.0.2.10	Kali Linux 2021	Attacker
10.0.2.100	Ubuntu Server 14.04	Mail server
10.0.2.101	Ubuntu Server 14.04	Mail server
10.0.2.200	Kali Linux 2021	Logger

Table 3.2: Description of the devices in the simulated network environment.

3.2 Proposed solution

Due to the nature of the problem and the available data, we have decided to develop an anomaly-based NIDS using semi-supervised techniques. The models employed are trained using data representing normal network behaviour (*normal dataset*), and then, used on the *attack dataset* to detect anomalous events belonging to a cyberattack.

We decided to develop two different semi-supervised models: an SPBN (or HSPBN) and a VAE, and compare their performance and explainability.

3.3 Methodology

Next, we describe the process followed to develop our system:

1. First, we decided to pre-process the raw network data using two alternative techniques that capture different information of the network behaviour, thus, obtaining two complementary pictures of the data provided.

2. The second step was to build and train an SPBN (or HSPBN) and a VAE for each of the two pre-processed normal datasets.

3. Having the normality models developed, we feed them with the attack data in order to detect malicious activity. The output of each model is a ranked list using the corresponding anomaly score in each case.

4. In order to fine-tune the results and discard false positives, we correlate for each

type of model the outputs using the two alternative pre-processed datasets for attack data.

5. Afterwards, for each of the suspicious data points we try to explain the type of cyberattack to which they belong. To do so, we identify the features that were relevant for the model to give a high anomaly score. Using the information of these explanations and applying clustering techniques, we group the anomalous points depending on their nature. In this way, points that are part of the same cyberattack are gathered.

6. Finally, we perform time series analysis of the most relevant features for each of the formed clusters, to refine the exact moment of the attack.

Figure 3.1 provides an overview of our network anomaly detection system. The components in the system are described in greater detail below.

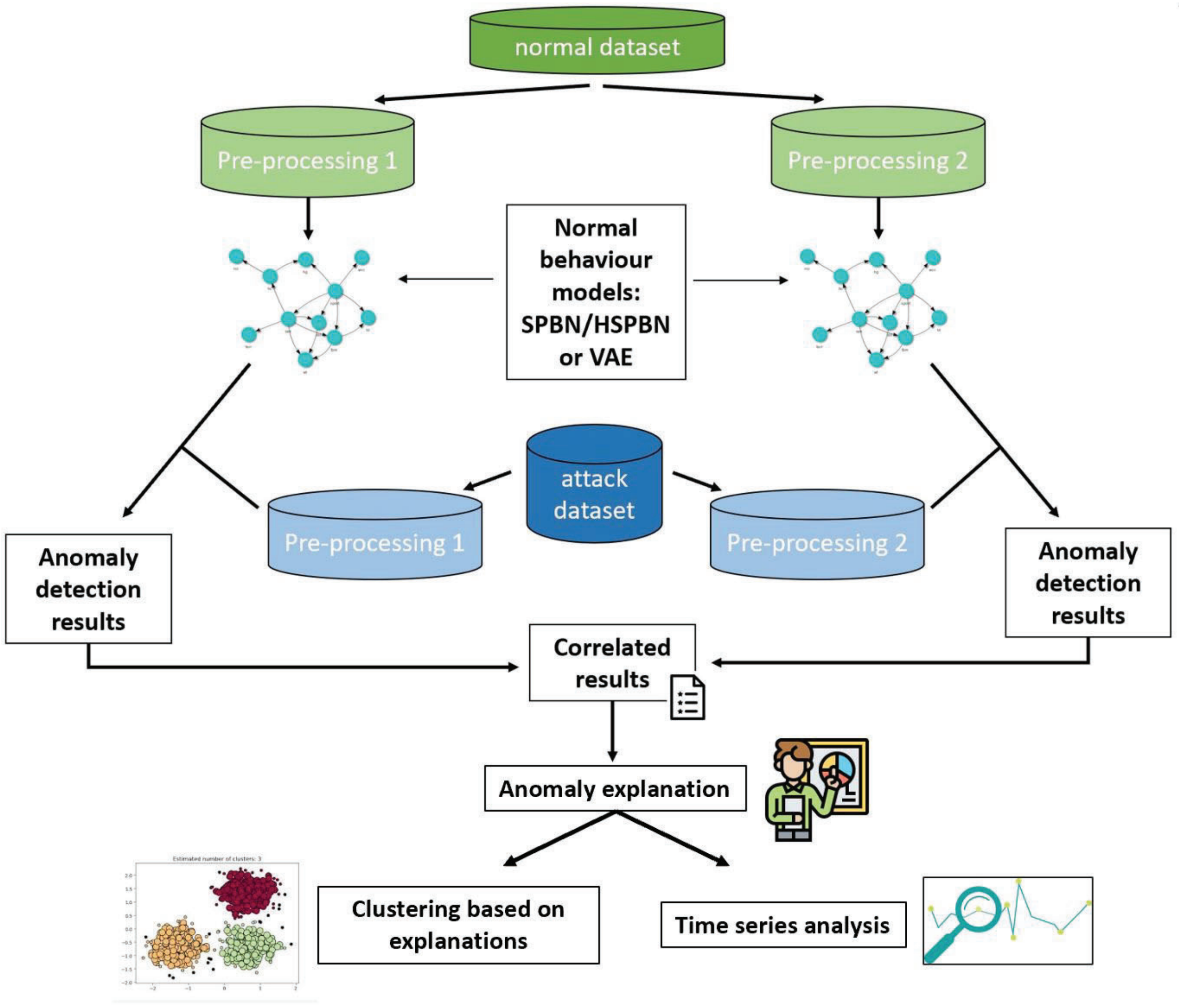


Figure 3.1: Developed network anomaly detection system.

3.3.1 Data pre-processing

As we have mentioned, we decided to pre-process the raw traffic data using two alternative methods. Thus, for each of the original datasets two different inputs for the semi-supervised models were obtained. The two different pre-processing shall be referred to as *connections* and *time_window*.

The first step in this phase is data parsing, to convert the data provided in the form of raw text data into a tabular format, obtaining the same structured information for each of the events in the datasets. Once this was done, we were able to analyse each of the packet's variables to understand their nature and range of possible values.

We then performed alternative processes for each desired output:

- ***Connections***.

  For this data pre-processing we need to remember the concept of connection in the context of the open systems interconnection (OSI) model [Day and Zimmermann, 1983]. The transport layer in the OSI model provides the functional and procedural means of transferring variable-length data sequences from a source host to a destination host from one application to another across a network, while maintaining the quality-of-service functions. Transport protocols may be connection-oriented or connectionless. In our original dataset, the two transport protocols used are TCP, which is connection-oriented, and UDP, which is connectionless. A TCP connection between client and server consists of three stages: connection establishment, data transfer and connection termination.

  In this pre-processing, we group together the events that belong to the same connection. This is straightforward in the case of TCP packets using TCP flags, that identify the different phases of a connection. In the case of events using UDP transport protocol, we define an UDP connection as all the events in the dataset with the same tuple (source IP address, destination IP address, source port, destination port).

  Hence, the resulting data points instances represent connections in the network. The variables describing each data point are displayed in Table 3.3.

  Although some variables like *sbytes*, *spkts*, *sttl* and *swin* can only take integer values, we consider them as continuous variables (as shown in Table 3.3) due to their large range, which would make it unfeasible to model them as discrete variables. Finally, continuous variables are standardized.

  Variables not marked in Table 3.3 as present in training are used to identify each data point and are not part of the model input. We will refer to them as the connection ID.

- ***Time_window***.

  In this case, we group the packet records into 2-minute non-overlapping sliding windows based on the source IP addresses to form aggregated features. This means that each of the resulting data point instance corresponds to the network statistics of a single source IP address within a 2-minute period. Note that such extraction allows us to identify the offending IP address and also the time window an anomaly belongs to, which are important for further analysis and

Variable name	**Type**	**Description**	**Present in training**
Stime	Datetime	Connection start datetime	
Ltime	Datetime	Connection end datetime	
ip_src	Categorical	Source IP address	
ip_dst	Categorical	Destination IP address	
dur	Continuous	Connection total duration	✓
sbytes	Continuous	Source to destination transaction bytes	✓
spkts	Continuous	Source to destination packet count	✓
sttl	Continuous	Source to destination time to live value	✓
swin	Continuous	Source TCP window advertisement value	✓
synack	Continuous	Time between SYN and SYN_ACK packets	✓
ackdat	Continuous	Time between SYN_ACK and ACK packets	✓
sload	Continuous	Source bits per second	✓
smeansz	Continuous	Mean of the flow packet size transmitted by the source	✓
dollar_payload	Binary	1 if the the payload of a packet in the connection contains the string '/ C$', "/ IPC$" or "/ ADMIN$"; 0 otherwise	✓
script_payload	Binary	1 if the payload of a packet in the connection contains the string "powershell.exe" or "This program cannot be run in DOS mode"; 0 otherwise	✓
buffer_overflow	Binary	1 if the most common byte in the payload is not the space; 0 otherwise	✓

Table 3.3: Output variables of the *connections* pre-processing.

decision making. The period of 2 minutes is chosen to balance between the practicality and quality of the aggregated statistics, where the statistics will be insignificant if the period is too short; while using a long time window means we cannot perform real time analysis. The cybersecurity expert also advised us on the selection of the time window value. Overall, we extracted aggregated features, which include:

- Mean (*frame_len_mean*) and standard deviation (*frame_len_std*) of bytes per event.
- Entropy of protocol type (*proto_entropy*).
- Entropy of destination IP addresses (*ip.dst_entropy*).
- Entropy of source (*srcport_entropy*) and destination ports (*dstport_entropy*).
- Entropy of TCP flags (*tcp.flag_entropy*).

The entropy for a discrete random variable X is defined as in information theory by the following formula:

$$H(X) = \sum_{i=1}^{n} \mathrm{P}(x_i) \log \mathrm{P}(x_i).$$

It is important to remark that the processed variables in this analysis are all continuous.

To ensure that meaningful statistics are captured, a data point that contains too few events, in our case less than 9, is removed from the processed dataset. This reduces noise in the training data, while not running the risk of losing meaningful information, as source IP adresses with very low activity are not suspected of cyberattacks. Finally, the statistics are normalized into Z-scores as input features for the models.

Likewise in the previous pre-processing, we need an identifier for each data point, that will allow us to determine the IP of the attacker as well as the time when the cyberattack was perpetrated. Here, the ID of an instance is composed by the source IP address and the start time of the 2-minute window to which the data point belongs.

3.3.2 Semiparametric Bayesian network model

Bayesian networks are a powerful tool for modelling the normal behaviour of the model. To select the appropriate type of Bayesian network, it is important to analyse the processed data. In the case of the *connections* pre-processing, we have both continuous and discrete variables, therefore a hybrid model is neccesary. In contrast, for the *time_window* pre-processing, we only have continuous variables. In both cases, we cannot claim that the continuous variables follow a Gaussian distribution, thus a SPBN and its hybrid version are the most suitable models.

Based on the theory presented in Section 3.3.2, we have developed the SPBN model for each of the normal data pre-processing. After testing the two structure learning algorithms previouly described, we decided to use the greedy hill-climbing (HC) algorithm guided by the cross-validated likelihood score for the final model; as the anomaly detection results were slightly better, and execution times were a bit shorter than using the PC algorithm. Then, we applied the parameter learning procedure proposed by Atienza (2021).

It is important to mention that we tested the HC training algorithms with different values for the patience parameter, λ. To compare the learnt structures, we calculated the structural Hamming distance [Tsamardinos *et al.*, 2006], that counts the number of arc additions, removals and reversals required to transform a DAG into another. Afterwards, we analysed their goodness to detect malicious data points and discussed the graph structure with the cybersecurity expert, marking those arcs that were unexpected for him. We also asked him about missing arcs that he expected to appear.

Since we can define a *white arc list* and a *black arc list*, with the arcs that we do and do not want to appear in the graph, respectively, we carried out further tests on the structure, with different combinations of arcs in these two lists based on the discussion with the expert. We compared the different structures and examined the performance of the resulting models on detecting anomalies, and chose the best one.

Finally, we would like to highlight that despite using a node type change operator in the implementation of the HC algorithm, all the nodes representing continuous variables turned out to have conditional KDE distributions.

Having the normal behaviour SPBN models for both pre-processing built and trained, we can detect suspicious events in the attack dataset. This is achieved by calculating the log-likelihood of each data point, as the log-likelihood value is a way to measure the goodness of fit for the model. The higher the value of the log-likelihood, the better the model fits a dataset. The log-likelihood value for a given model can range from negative infinity to positive infinity. The log-likelihood of the model for a particular instance is negative infinity, if the probability of that instance being generated by the model is zero. Therefore, the value of the log-likelihood can be seen as an anomaly score: the lower the log-likelihood, the higher the score and more anomalous.

3.3.3 Variational autoencoder model

For the development of the VAE model, we first had to determine its architecture. We set the size of the latent representation layer to be 25 for the *connections* pre-processing and 15 for the *time_window* pre-processing. In addition, the encoder and the decoder each has three hidden layers with size 128, 128, and 256, respectively, as illustrated in Figure 3.2. In the figure, nodes that are coloured in black represent the observed data (used as both inputs and outputs), while the unshaded nodes are unobserved latent variables corresponding to the hidden layers. The exact size or dimension of the layers, which is shared by the models for both pre-processing, is shown above the nodes. It is important to note that to select the optimal size of the hidden layers, we performed grid search of sizes power of two for the encoder and decoder using a symmetrical pattern, whereas for the latent representation layer, we bounded the size between 0.5 and 2.5 times the size of the input layer, as there is no well-established rule in the literature for this, but the larger it gets the more likely it is that overfitting will occur.

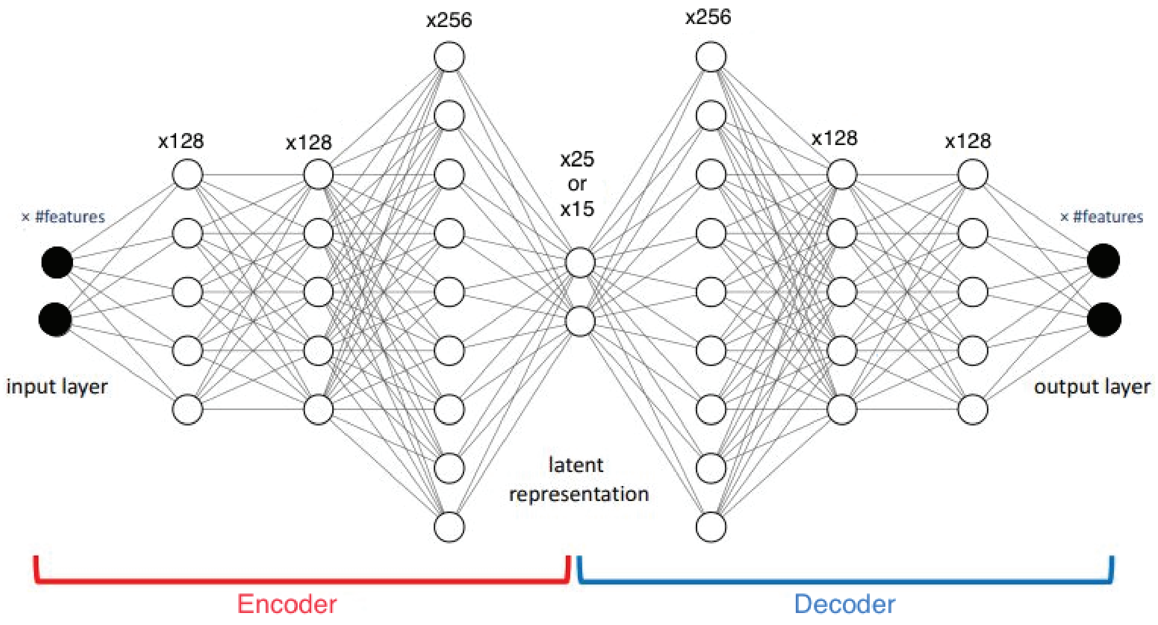


Figure 3.2: Variational Autoencoder architecture.

The training algorithm of the VAE was implemented using TensorFlow, which provides powerful tools of automatic differentiation and comes with built-in optimization routines.

We now present only a brief outline of the training of the VAE due to its complexity. Before starting the training procedure, the parameters in the VAE are randomly initialized. This subsequently allows us to perform a forward pass on the encoder by computing the distribution of the latent representation layer. With this, several samples can be generated from the distribution which are used to compute the variational lower bound, $\mathcal{L}$, which consists of a Kullback–Leibler (KL) divergence term and an expectation term, as shown in the following equation:

$$\mathcal{L} = -D_{KL}[q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})] + \mathbb{E}_q[\log p(\mathbf{z}|\mathbf{x})], \tag{3.1}$$

where $\mathbf{z}$ is the latent representation of the input features $\mathbf{x}$. The distribution $p()$ corresponds to the prior and conditional distribution of the VAE model; while $q()$ is a variational approximation [Blei *et al.*, 2016] of $p()$, generally chosen to be Gaussian. Luckily, this objective function can be maximized with stochastic optimization techniques since the gradients are readily available via automatic differentiation. To do so, we employed Adam as the optimization algorithm, which enables training in minibatches. Furthermore, by choosing a small minibatch size and discarding the data after one epoch, we can reduce the training time and the computational burden. We would like to highlight that the IDs of the data points are not used at all during training.

Finally, to achieve a good performance, we needed to fine-tune the hyper-parameters of the VAE model for both pre-processing. Hyper-parameters include learning rate, size of the minibatch, regularization parameter, momentum and number of epochs. We performed several tests to determine the best values for them.

Once the parameters are optimized after training, the VAE model is used for anomaly detection, where a data point identified by its ID is recognized as malicious when the reconstruction error of its input features is high.

The reconstruction error can be calculated using the mean square difference (or mean square error, MSE) between the observed features and the expectation of their reconstruction as given by the output of the VAE. A high reconstruction error is generally observed when the network behaviour differs greatly from the normal behaviour that was learnt by the VAE. So, we will consider the reconstruction error as a measure of the anomaly score.

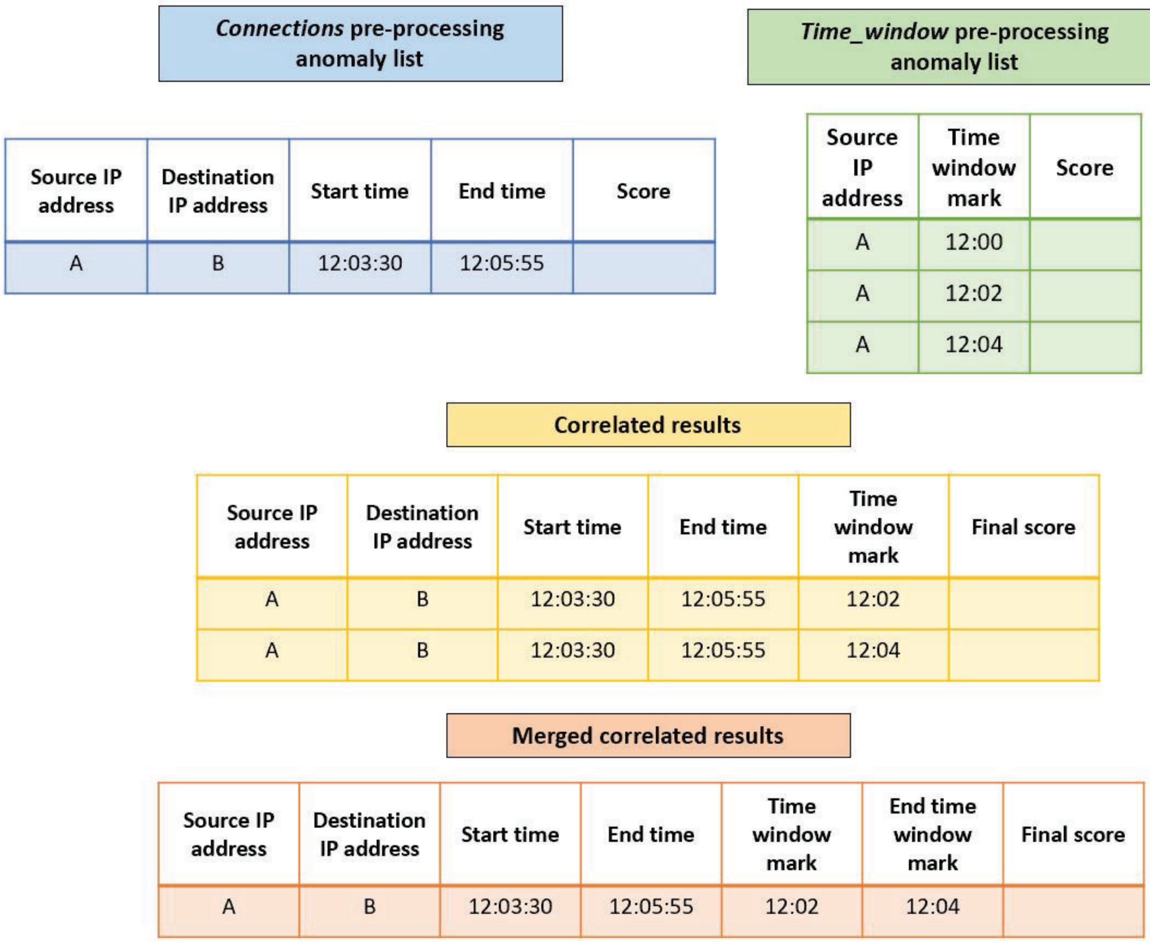
3.3.4 Anomaly detection results correlation

After calculating the anomaly score (log-likelihood for SPBN and MSE for VAE) for data points from both pre-processing (*connections* and *time_window*) of the *attack* dataset, we correlated the two lists of anomalous instances with the objective of fine-tuning the results and discarding false alarms.

However, in order to shorten the lists before correlating results and try to keep only suspicious events, we had to set a threshold. So, we decided to just preserve those instances whose score was above the 75th percentile of the scores for each of the lists, after testing several values.

Each list of malicious points contains the corresponding data point ID and its anomaly score. Recall that the ID for the *connections* pre-processing is composed of the source and destination IP addresses of the connection along with its start and end time. Meanwhile, the data point ID in the *time_window* pre-processing consists of the source IP address and the start time of the 2-minute window to which the data point belongs, that we will refer to as *time window mark*.

So for the linkage of both malicious point lists, we link each connection (data point from the *connections* pre-processing) with all the data points from the *time_window* pre-processing with the same source IP address, if the 2-minute window to which the data point from the *time_window* pre-processing belongs intersects the time interval between the start and end time of the connection. For the shake of simplicity, we illustrate in Figure 3.3 how the linkage was done.



Connections pre-processing anomaly list

Source IP address	Destination IP address	Start time	End time	Score
A	B	12:03:30	12:05:55	

Time_window pre-processing anomaly list

Source IP address	Time window mark	Score
A	12:00	
A	12:02	
A	12:04	

Correlated results

Source IP address	Destination IP address	Start time	End time	Time window mark	Final score
A	B	12:03:30	12:05:55	12:02	
A	B	12:03:30	12:05:55	12:04	

Merged correlated results

Source IP address	Destination IP address	Start time	End time	Time window mark	End time window mark	Final score
A	B	12:03:30	12:05:55	12:02	12:04	

Figure 3.3: Example of correlation of results.

It is important to highlight, that the time interval of most of the connections lies within a unique 2-minute time window, thus only one entry is added for those connections in the correlated results list. For less than 10% of the total number of connections, more than one entry is added.

Once we have done the linkage, we had to calculate the final score for each of the entries in the correlated results list. To obtain that final value, we tried two alternatives: on the one hand, normalising (min-max scaling) the scores of each of the original tables and averaging, and on the other hand, standardasing (Z-score normalization) and averaging. After trying both possibilities and comparing the results for anomaly detection, we chose the latter.

In cases where we had several entries in the final list for the same original connection, we merged all of them into a single one. The value for the *time window mark* column for the resulting entry is chosen to be the lowest one among the merged instances, and the final score value is the mean of the merged scores. Besides, we added another column to the final merged list of correlated results named *end time window mark*, whose value is set to be the start time of the 2-minute window to which the end time of the connection belongs.

Finally, we ranked the instances of the merged correlated results list and joined contiguous entries who share the same value for the source IP address, *time window mark* and *end time window mark* columns and make a list with the destination IP addresses of the joined connections. Our main interest is identifying the attacker, as it may be attacking several devices at the same time. We could go even forward and delete entries in the final merged list in case there is a previous entry that has the same values for all the colums identifying each item of the list.

With this final step, we could fear losing the exact time the attack was perpetrated. Nevertheless, as we will analyse the time series of suspicious IPs later, this is not a problem. By doing this, the final merged correlated results list does not have too many entries, thus we do not need to set another threshold to reduce it. This is particularly important, as we do not want the cybersecurity analyst to be overwhelmed by tons of alarms.

3.3.5 Anomaly explanation

While most existing anomaly detection works in the literature consider only the evaluation on detection accuracy, we go beyond and provide an explanation on why a data point is flagged as anomalous.

The proposed solution is to identify the features that are most relevant for the particular model to mark a data point as an anomaly. That is, in the case of the SPBN model, the variables from the data point that contribute the most to the negativity of the log-likelihood value. For the VAE model, the features that add the most to the variational lower bound (Equation (3.1)). This is significant since it challenges the belief that the decisions of artificial intelligence models cannot be interpreted.

For achieving our goal, we analysed the partial derivative for each feature of the data point with respect to the objective function, that for the SPBN model is the log-likelihood, while for the VAE model is the variational lower bound. These partial derivatives were obtained straightforward by automatic differentiation for both models. In doing so, we tried to approach the following question: How does the objective function vary if a feature in the anomalous data point increases or decreases by a small amount?

The intuitive reasoning underlying our idea is that, given the trained model and a suspicious data point, if the objective function changes a lot when a particular feature of the anomalous instance is varied by a small amount, then this feature at its current value is considered to be significantly abnormal.

Explanations obtained from the partial derivatives, are not only useful on their own, but have further applications. For example, even without having the ground truth labels in the attack dataset, the flagged anomalies can be clustered based on their gradients into groups that share similar behaviour, making it easier, if necessary, for cybersecurity analysts to investigate further and eventually label them.

This is of particular interest given that some cyberattacks are constituted by several data points rather than a single one; even though there may be one event that is more significant in raising the alarm of the attack and has a higher anomaly score among the points in the same cluster. So, the idea is that if the clustering is effective, attacks should be limited to a relatively small number of clusters (as we will see below that in the attack dataset we only have two different types of cyberattacks).

To perform the clustering process, we first obtained from each data point a normalised vector whose components are the values of the partial derivatives of its features. Then, we applied different clustering methods and analysed their performance in order to select the best one.

The clustering techniques that we tested in our results were:

- **Partitional clustering: k-means.**

  Partitional clustering makes natural divisions of the data into a fixed number of clusters, that has to be determined a priori. k-means algorithm [Forgy, 1965] is one of the fastest clustering algorithms available. It aims to partition the N observations into k sets $\{S_1, \ldots, S_k\}$, so as to minimize the within-cluster sum of squares. Formally, the objective is to find:

  $$\text{argmin} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} ||\mathbf{x} - \boldsymbol{\mu_i}||^2,$$

  where $\boldsymbol{\mu_i}$ is the mean of the i-th cluster or centroid. The algorithm uses an iterative refinement technique that consists of two steps: first, assign each observation to the cluster with the nearest mean and then, update the clusters' centroids. This is repeated until convergence. Nevertheless, it is not guaranteed to find the optimum partition, thus falling in local minima. Therefore, k-means algorithm is very sensitive to the initial partition.

  To try to solve this, we restart the algorithm with different random seeds, and also, taking as input the result of applying hierarchical clustering. Besides, we also tested the k-means++ initialisation algorithm [Arthur and Vassilvitskii, 2007], the exact execution of which is as follows: first, it chooses one center uniformly at random among the data points. Secondly, for each data point $\mathbf{x}$ not chosen yet, it computes $D(\mathbf{x})$, the distance between $\mathbf{x}$ and the nearest center that has already been chosen. Then, it chooses one new data point at random as a new center, using a weighted probability distribution where a point $\mathbf{x}$ is chosen with probability proportional to $D(\mathbf{x})^2$. Finally, it repeats the two last steps until k centers have been selected.

  In addition, we also have to take care of determining the value of the k parameter. In the case of *Titanium*'s dataset this is an easy task as the attack dataset is very small and we now beforehand the number of attacks. Nevertheless, in a general situation this will not be the case and we have to automatise the selection of the k parameter. To do so, we must take into account the nature of our data.

  As we have already mentioned, consecutive anomalies during a particular time interval usually belong to the same cyberattack. That is, since our system will run locally, it would be unlikely that several types of attacks would be performed at the same time in a specific part of the whole network. Therefore, anomalies that are part of the same cluster will come in a sequence. It is true that a certain type of attack could be repeated over time, and points will be assigned to the same cluster, but still that new set of events will form a sequence.

This could remind us of the notion of *recurring concept drift*, where data instances change between two or more statistical characteristics (which in our case are representative of attack classes) several times. Neither of the classes disappears permanently but all of them arrive in turns. A data stream containing a recurring concept drift might look like as follows, where different colors indicate different classes:

$$S = \{\dots, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9, \mathbf{x}_{10}, \mathbf{x}_{11}, \mathbf{x}_{12}, \mathbf{x}_{13}, \mathbf{x}_{14}, \mathbf{x}_{15} \dots\}.$$

To deal with this situation, we will first sort the final list of anomalous data points in time and then apply the adaptive stream k-means algorithm, proposed by Puschmann *et al.* (2017), which is a partitioning-based data stream clustering algorithm. Although, in general, partitioning-based clustering algorithms need k as an input parameter, and have difficulties to adapt concept drifts in the input data, Puschmann *et al.* (2017) claim to overcome these two main problems.

The algorithm is composed of two main phases, which are the *initialization phase* and the *continuous clustering phase*. In the initialization phase, m data instances are accumulated. Then groups of candidate centroids are determined. In order to find k and determine candidate centroids, the PDF of each variable of the data is calculated using KDE. All directional changes in the shape of PDF curve are accepted as signs of beginning of a new region. Here the region can be defined as the area between two consecutive directional changes of the PDF curve. The number of regions is considered as a candidate k and the centers of these regions are considered as candidate initial centroids. This process is pursued for each feature of the data separately. Because different features generally show different distributions, then more than one k values, and different candidate centroids are found. After finding all candidate k values, clustering is performed for a set of k values where $k \in [k_{min}, k_{min} + k_{max}]$. Clustering results of different k values are compared according to silhouette coefficient, and the best k is selected with its corresponding centroids. Then, during the continuous clustering phase, concept drift detection is performed. If no concept drift occurs, clustering of the input data proceeds. However if a concept drift exists, the parameter k and the centroids are recalculated (that is, the algorithm is re-initialized) before clustering continues with a new k and centroids. For concept drift detection, standard deviation and mean of the input data are stored during the execution. The algorithm tracks how these two values change over time and predicts a concept drift according to the change.

- **Probabilistic clustering: Gaussian mixture models**.

  Probabilistic clustering relies on the hypothesis that data come from a mixture of k probability distributions, one for each cluster. Observations are probabilistically assigned to clusters, rather than deterministically.

  In particular, we tested Gaussian mixture models. To find the mean and variance of each cluster, the expectation-maximization algorithm [Laird, 1993] is used, which is a form of optimization function. The expectation-maximization iteration alternates between performing an expectation step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters, and a maximization step, which computes parameters maximizing the expected log-likelihood of a sample belonging to the cluster distribution found on the expectation step. These parameter-estimates are then used to determine the distribution of the latent variables in the next estimation step.

This option did not give optimal results, the reason being that data do not fit a mixture of Gaussian distributions. We corroborated this by performing statistical tests for normality on the two clusters obtained with the partitional method using Royston's test.

Finally, we calculated the mean of the partial derivatives of each feature within each final cluster. This would serve to explain each type of cyberattack. Thus being a distinguising mark of the attack, this could also be useful for identifying future attacks of the same nature.

3.3.6 Time series analysis

The last step of our system is to perform time series analysis of the most relevant features for each of the clusters we obtained in the previous step. We also examined the time series for the number of bytes, which is the most general feature but provides valuable information for detection.

The time series analysis is only performed for the most relevant source and destination IP addresses that constitute the ID of an anomalous connection for the semi-supervised detection model, being the source IP address the suspected attacker.

We tried three different methods for the time series analysis including:

- Deep neural networks, in particular, LSTM: using them for time series forecasting and comparing the predictions with the observed time series, flagging as malicious those points whose prediction deviates greatly from the ground truth. LSTM are particularly good for this purpose, as they are memory-based models.
- Isolation forest [Fei *et al.*, 2008] explicitly identifies anomalies instead of profiling normal data points. Isolation forest, like any tree ensemble method, is based on decision trees. In other words, isolation forest detects anomalies purely based on the fact that anomalies are data points that are few and different. The anomalies isolation is implemented without employing any distance or density measure. However, its main drawback is that we have to set a parameter with the percentage of anomalous points in the series, which is unknown beforehand.
- Statistical techniques to detect anomalies: we performed time series decomposition applying the seasonal-trend decomposition using locally weighted scatterplot smoothing (STL) method, which uses locally fitted regression models to decompose a time series into trend, seasonal, and remainder or residual components. Besides, we used Pearson's correlation coefficient between the time series of source and destination IP addresses for the attack dataset. But also, between each of them and their corresponding time series for the normal behaviour by adjusting the possible "lag". To do this, we found the periodicity of the normal

behaviour time series and calculated Pearson's correlation coefficient between the normal time series and the shifted attack one with a maximum shift equal to the period of the former. The selected "lag" corresponded to the first maximum of the correlation coefficient values obtained.

The option finally chosen for the implementation of our IDS was the latter, that is, statistical techniques. In the following section, we show exactly how the analysis was done and now the exact times of the anomalies were obtained for *Titanium*'s attack data.

3.4 Implementation

For the development of our system, we have employed Python language in the Google Colab environment. To learn the SPBN models, we used PyBNesian library [Atienza *et al.*, 2022b]; whereas for the training of the VAE we used the TensorFlow framework, which is highly parallelizable and thus, runs very quickly. Although, we used these libraries, we had to make some modifications to adapt them to our data, and we also performed a high number of test to achieve good performance.

We ourselves implemented in Python the pre-processing as well as correlation of results to obtain the final list of anomalies. As mentioned above, the partial derivative analysis to obtain individual explanations was implemented using TensorFlow. The adaptive stream k-means algorithm for the clustering of explanations was implemented in Python with the help of several existing libraries to detect concept drift. Finally, the statistical analysis of time series was implemented by us, making use of available functions, for example to calculate Pearson's correlation coefficient.

Chapter 4

Results

In this section, we show the capabilities of our system to detect anomalies and generate explanations. The proposed system was tested on the data described in Section 3.1. However, as we have already mentioned, these data are not properly labelled, which makes it difficult to obtain a quantitative value of the system's performance.

4.1 *Titanium*'s datasets

The first step was to pre-process both the *normal* and *attack* datasets using the two alternatives described in Section 3.3.1. Then, we trained a semi-supervised model for each of the two pre-processed normal datasets.

In the case of using SPBNs, we tried different values for the patience parameter, $\lambda \in \{5, 10, 15, 20, 25\}$. Values outside that range gave very poor results in terms of anomaly detection. It is also important to highlight that for the *connections* pre-processing, graph structures varied for different values of the patience parameter, unlike the case of the *time_window* pre-processing. Table 4.1 shows the structural Hamming distance between the graph structures for different values of λ. After, evaluating the anomaly detection results for each possible structure, we selected to take $\lambda = 10$.

λ	**5**	**10**	**15**	**20**	**25**
5		7	6	7	6
10			4	3	6
15				1	5
20					5
25					

Table 4.1: Structural Hamming distance between the different HSPBN structures for the *connections* pre-processing.

As we mentioned above, we also executed the HC structure learning algorithm keeping some arcs fixed and forbidding others, after discussion with the cybersecurity expert.

The final graph structures of the normal behaviour models for the *connections* and *time_window* pre-processing are shown in Figures 4.1 and 4.2, respectively. It is important to mention that in the HSPBN graph structure for *connections* pre-processing (Figure 4.1), only ten variables (instead of twelve), as two of them are independent from the rest. Furthemore, we would like to point out that, for example, by banning the arc from *swin* to *sttl* which was unintuitive for the cybersecurity expert, the performance of the system degrades a lot.

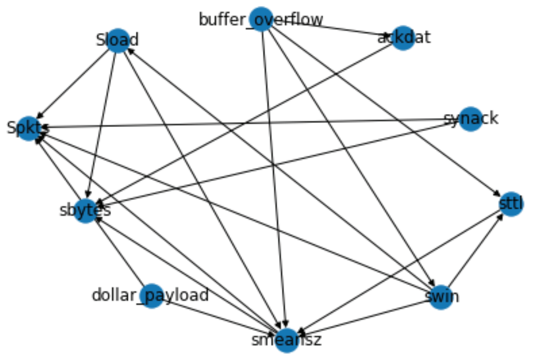


Figure 4.1: HSPBN graph structure for *connections* pre-processing.

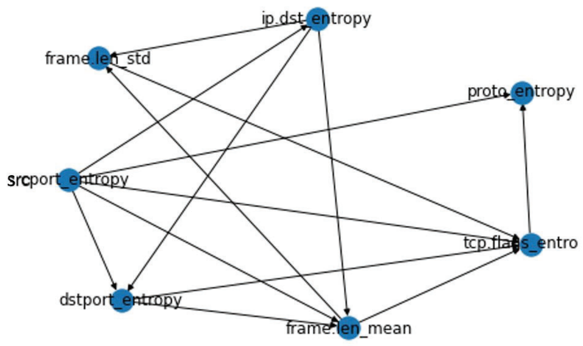


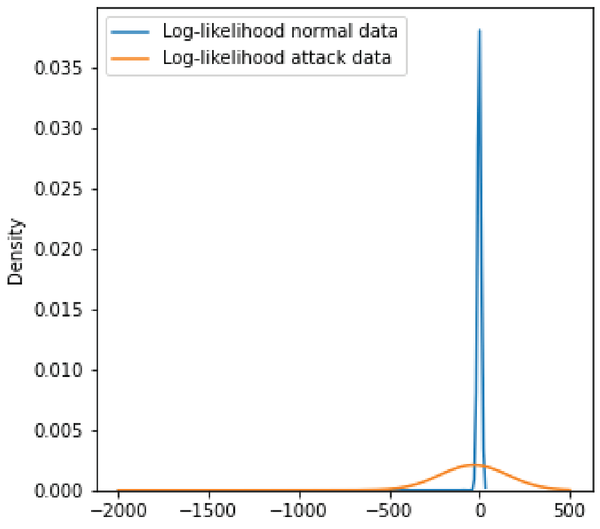
Figure 4.2: SPBN graph structure for *time_window* pre-processing.

Despite that in the *attack* dataset we also have normal background behaviour, there is a notable difference between the mean log-likelihood of *normal* and *attack* datasets. Exact values are presented in Table 4.2. Besides, Figures 4.3(a) and 4.3(b) represent the log-likelihood distribution output yielded by the SPBN models for the *connections* and *time_window* pre-processing, respectively, for both *normal* and *attack* datasets.

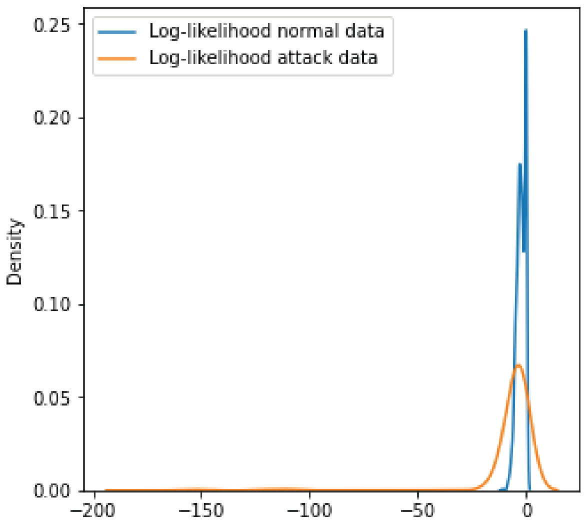
	Pre-processing	
	Connections	***Time_window***
Normal dataset	-0.77	-2.34
Attack dataset	-164.47	-7.35

Table 4.2: Mean log-likelihood values for the *connections* and *time_window* pre-processing SPBN models.

The anomaly detection results on the *attack* dataset after correlating the outputs of



(a) *Connections* pre-processing.



(b) *Time_window* pre-processing.

Figure 4.3: Log-likelihood probability density for *normal* and *attack* datasets.

both pre-processing are displayed in Table 4.3. We would like to remind that each entry of the table is a combination of several individual connections.

	Source IP address	**Time window mark**	**End time window mark**	**Destination IP addresses**
1	10.0.2.10	24/02/2022 15:00	24/02/2022 15:02	10.0.2.5
2	10.0.2.10	24/02/2022 13:40	24/02/2022 13:40	10.0.2.5 10.0.2.7
3	10.0.2.10	24/02/2022 15:02	24/02/2022 15:02	10.0.2.5
4	10.0.2.10	24/02/2022 14:52	24/02/2022 14:52	10.0.2.5
5	10.0.2.5	24/02/2022 13:38	24/02/2022 13:38	10.0.2.10
6	10.0.2.7	24/02/2022 13:38	24/02/2022 13:38	10.0.2.10
7	10.0.2.10	24/02/2022 14:52	24/02/2022 14:52	10.0.2.5 10.0.2.7
8	10.0.2.10	24/02/2022 13:38	24/02/2022 13:38	10.0.2.5 10.0.2.7 10.0.2.100 10.0.2.101 10.0.2.200
9	10.0.2.10	24/02/2022 15:00	24/02/2022 15:00	10.0.2.5

Table 4.3: Final merged list of anomaly detection results for the SPBN model.

Looking closely at the final anomaly detection list, we could confirm that the SPBN model captures the two cyberattacks: port scanning and lateral movement. This was done with the help of the cybersecurity expert who verified that the anomalies detected corresponded to the simulated attacks.

On the one hand, the port scanning attack, that occurs between 13:38 and 13:42 and it is performed by the IP address 10.0.2.10 to the IP addresses 10.0.2.5, 10.0.2.7, 10.0.2.100, 10.0.2.101 and 10.0.2.200. It corresponds to lines 2 and 8 of Table 4.3.

The SPBN models also flag as suspicious the answer connections from 10.0.2.5 and 10.0.2.7 to 10.0.2.10 (lines 5 and 6), which could be seen as a false alarm, although it is not very disturbing, so it could be considered a minor error.

On the other hand, it captures the lateral movement attack which is perpetrated around 14:52 to 10.0.2.5 and 10.0.2.7 (lines 4 and 7). The SPBN model also signals an anomalous connection from 10.0.2.10 to 10.0.2.5 following up the main attack, where the attacker is trying to access further information (lines 1, 3 and 9).

It is important to highlight that false positives are reduced by almost 100% thanks to the correlation step, as there are significantly more false positives in the separate anomaly result lists. Futhermore, this option is more convenient than setting a threshold for one of the SPBN models, as deciding its value can be quite difficult and adapting it to a particular network environment is not a straightforward task. We may even have to design an algorithm so that the value is adapted periodically.

As for the VAE, the exact architecture was specified in Section 3.3.3. The number of trainable parameters in the case of *connections* pre-processing is 121K, whereas for the *time_window* pre-processing is 119K. Execution times, once the fine-tune the hyper-parameters was done, were under 30 seconds as both datasets are relatively small.

In terms of the reconstrucion errors of the *normal* and *attack* datasets, the difference between their mean MSE values is smaller than in the case of the SPBN model. This can be observed in Table 4.4 and in Figures 4.4(a) and 4.4(b), which illustrate the reconstruction error or MSE distribution of the *connections* and *time_window* pre-processing, respectively.

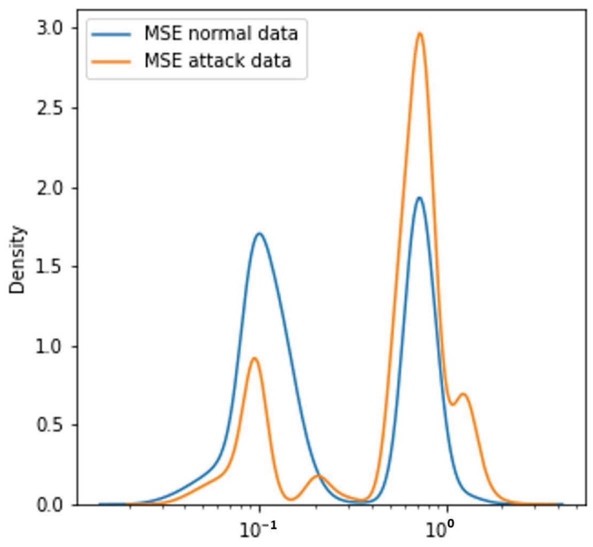
	Pre-processing	
	Connections	***Time_window***
Normal dataset	2.25	0.98
Attack dataset	3.19	1.56

Table 4.4: Reconstruction errors for both VAE models.

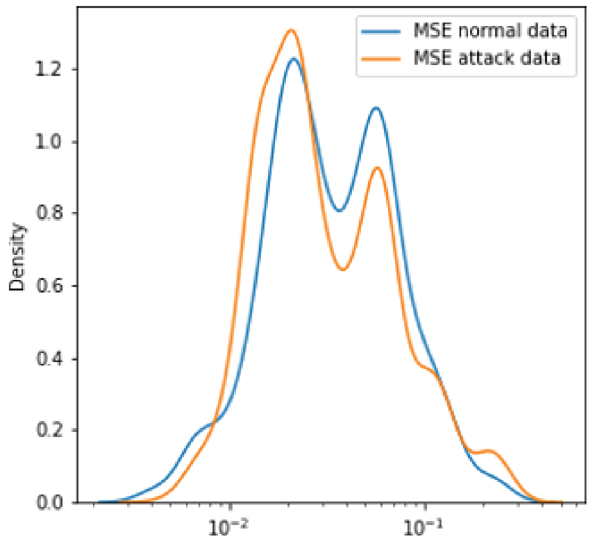
The final results of anomaly detection, after linking the outcomes of the two pre-processing, can be seen in Table 4.5. Likewise the SPBN model, the VAE model also captures both the port scanning attack to the five IP addresses: 10.0.2.5, 10.0.2.7, 10.0.2.100, 10.0.2.101 and 10.0.2.200, and the lateral movement to the victim IP addresses 10.0.2.5 and 10.0.2.7. In this case, the lateral movement attack, which is the most dangerous as the attacker tries to get access to private information, is detected with higher priority compared to the port scanning attack.

Furthermore, only one entry of the table, number 2, can be considered a false positive. Thus, we could claim that false alarms have been almost completely eliminated from the list of Table 4.5 that is provided to the end-user of our system.

Although both models have successfully detected both attacks with a minimum false alarm rate, the SPBN model can provide more information regarding interpretability of the model, in case we want to make further queries. Furthermore, it can better diferentiate normal behaviour from anomalous one compared to the VAE model, as we have shown in Table 4.2 and Figure 4.3.



(a) *Connections* pre-processing.



(b) *Time_window* pre-processing.

Figure 4.4: Mean reconstruction error probability density for *normal* and *attack* data after *connections* pre-processing.

	Source IP address	**Time window mark**	**End time window mark**	**Destination IP addresses**
1	10.0.2.10	24/02/2022 14:52	24/02/2022 14:52	10.0.2.5 10.0.2.7
2	10.0.2.5	24/02/2022 14:52	24/02/2022 14:52	10.0.2.10
3	10.0.2.10	24/02/2022 15:02	24/02/2022 15:02	10.0.2.5
4	10.0.2.10	24/02/2022 13:38	24/02/2022 13:38	10.0.2.7
5	10.0.2.10	24/02/2022 15:00	24/02/2022 15:00	10.0.2.5
6	10.0.2.10	24/02/2022 13:38	24/02/2022 13:38	10.0.2.101
7	10.0.2.10	24/02/2022 13:40	24/02/2022 13:40	10.0.2.5
8	10.0.2.10	24/02/2022 15:00	24/02/2022 15:02	10.0.2.5
9	10.0.2.10	24/02/2022 13:38	24/02/2022 13:38	10.0.2.5 10.0.2.100
10	10.0.2.10	24/02/2022 13:40	24/02/2022 13:40	10.0.2.7 10.0.2.100 10.0.2.101 10.0.2.200
11	10.0.2.10	24/02/2022 13:38	24/02/2022 13:38	10.0.2.200

Table 4.5: Final merged list of anomaly detection results for the VAE model.

We now move on to analysing the explanations automatically generated by our system. The system obtains individual explanations for each of the anomalous connections. Figures 4.5 and 4.7 are examples of the type of explanations provided for two instances from the *connections* pre-processing, while 4.6 and 4.8 show the explanations of their associated data points from the *time_window* pre-processing.

The suspicious connection represented in Figure 4.5 is a port scanning attack event from 10.0.2.10 to 10.0.2.5 between 13:38:50.43 and 13:38:51.42. We note that the *swin* (source to destination advertisement values) and *buffer_overflow* features seem to be the most relevant for the system to flag the connection as anomalous. Nevertheless, the expert disagreed with this, as he considered that the *dur* variable would have been more reasonable, as the connections belonging to the port scanning attack are usually very fast. When analysing the partial derivatives of the event from the *time_window* pre-processing to which it its linked, we observe that the partial derivative for the entropy of destination ports and of destination IP addresses variables have very large values, as shown in Figure 4.6. This confirmed what the cybersecurity experts expected from this type of attack.

The connection of Figure 4.7 is made from 10.0.2.10 to 10.0.2.5 between 14:52:01.58 and 14:52:01.64. It corresponds to the lateral movement attack. We can observe that there is a clear winning feature, *dollar_payload*, to mark the connection as abnormal. This is because the C$, IPC$ or ADMIN$ exploit (which is represented by the *dollar_payload* variable) allows the attacker to stealthily gain control of the victim machine. It is important to note that there is no one-to-one correspondence between the presence of this variable and cyberattack, however it gives important clues to the cybersecurity expert when analysing the explanations provided by our system. For its associated data point from the *time_window* pre-processing, depicted in Figure 4.8, the partial derivative of the *srcport_entropy* variable takes a relatively large value in comparison with the others. Nevertheless, the cybersecurity expert states that this latter explanation is not particularly meaningful for this type of attack.

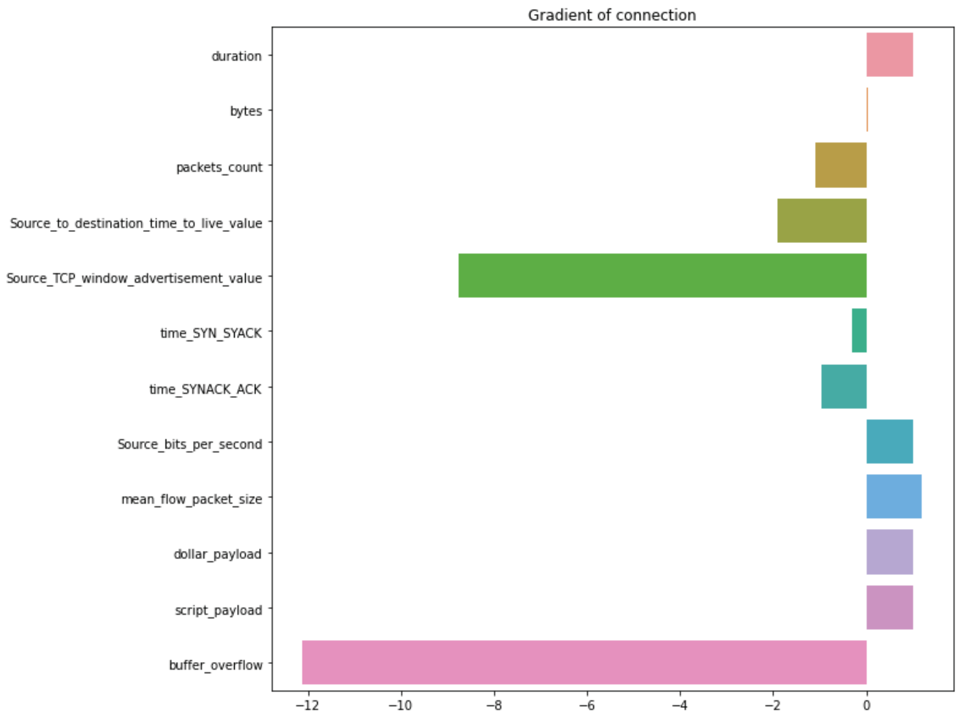


Figure 4.5: Explanations for the *connections* pre-processing based on partial derivatives for a data point belonging to the port scanning attack with ID: 10.0.2.10 (source), 10.0.2.7 (destination), 13:38:50.43, 13:38:51.42.

After normalising the vectors of the individuals' explanations based on partial derivatives, we performed clustering. As we mentioned in Section 3.3.5, we applied several techniques and finally decided to use the adaptive k-means algorithm for our system.

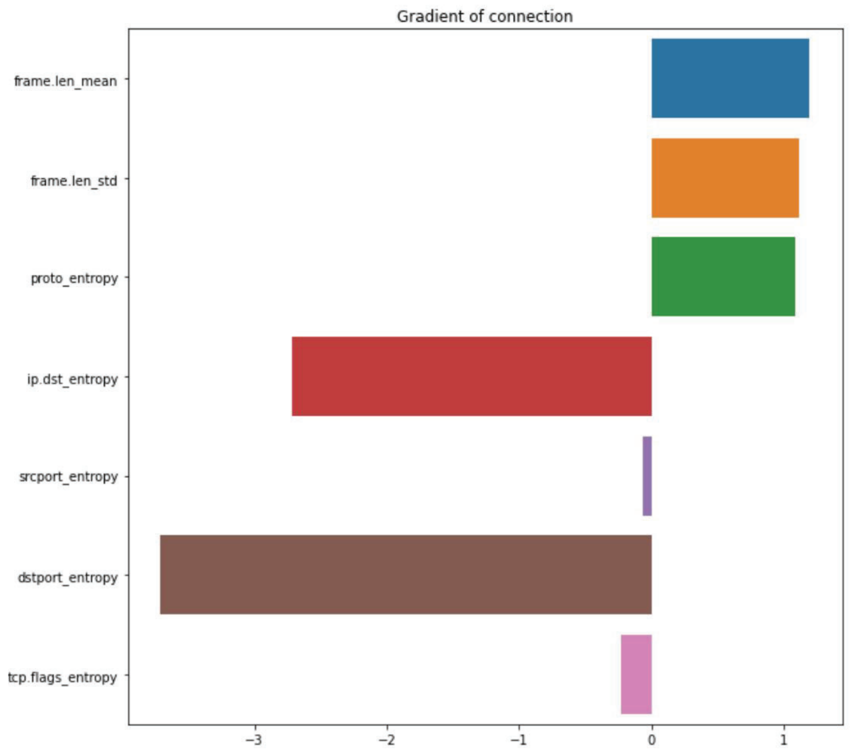


Figure 4.6: Explanations for the *time_window* pre-processing based on partial derivatives for a data point belonging to the port scanning attack with ID: 10.0.2.10, 13:38.

When using the adaptive k-means algorithm, we first have to order the explanation vectors of anomalous events over time. Then, we tried two alternatives: considering all these data as one stream or splitting the data into two parts with the same duration in time, so that the port scanning attack and the lateral movement attack are in different data streams.

For both alternatives, the resulting number of clusters is two, although there are some points that are assigned to the wrong cluster. Overall, we could claim that the generated explanations are characteristic of the type of attack to which the connection belongs and can distinguish them from others. Furthermore, it is important to remark that in the case of using two separate data streams, the algorithm is able to detect concept drift and add another cluster.

We plot in Figure 4.9 how data points belonging to each attack are distributed. The X-axis represents the partial derivative for the normalized *dollar_payload* variable, while the Y-axis represents the partial derivative for the normalized *swin* variable. The data points are coloured red and blue for the port scanning and lateral movement attacks, respectively. Blue points that overlap with red ones are not correctly assigned to their ground truth cluster by the clustering algorithm. However, these only represent 7.07% of the total number of anomalous points detected by our algorithm.

We also calculated the average explanations for each of the two clusters, which are shown in Figures 4.10 and 4.11, respectively. The black bars in the figures reflect the standard error for the partial derivatives, which are useful for assessing the significance of the partial derivatives, that is, whether it is due to noise or not. We observe in both figures that only a small subset of the features have large gradients

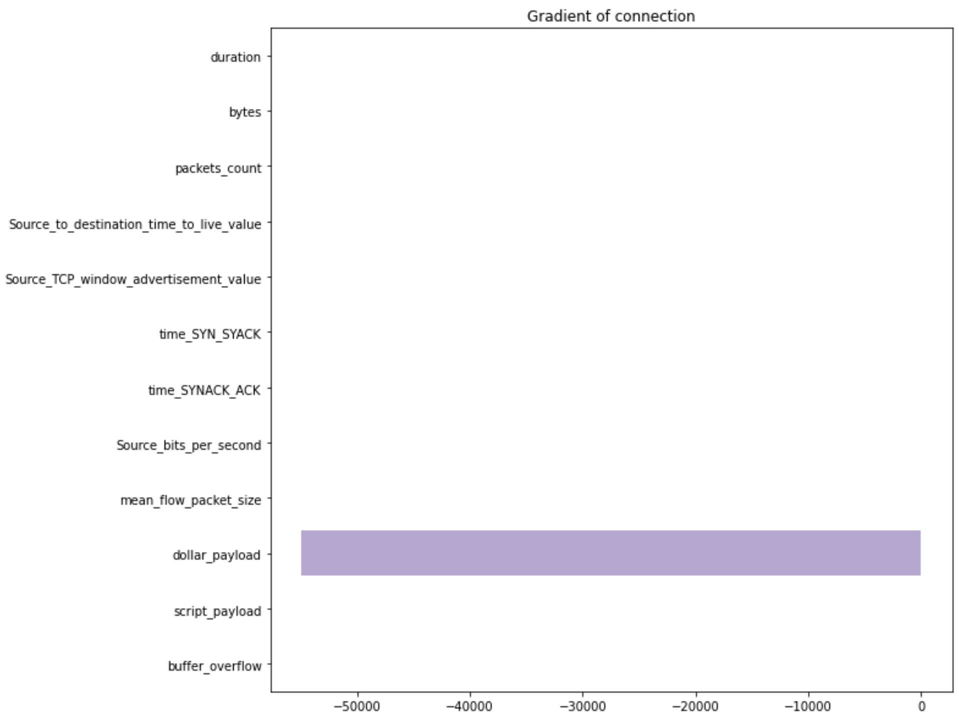


Figure 4.7: Explanations for the *connections* pre-processing based on partial derivatives for a data point belonging to the lateral movement attack ID: 10.0.2.10 (source), 10.0.2.5 (destination),14:52:01.58, 14:52:01.64.

with low standard errors, which are *swin* and *buffer_overflow* for Figure 4.10 and *dollar_payload* for Figure 4.11. This is in agreement with what we observed in the examples of individual explanations.

Finally, we performed time series analysis using the information from the final merged list of anomaly detection results, studying the time series of the indicated IP addresses.

As we mentioned earlier, we decided to apply statistical techniques for this analysis. We discarded LSTM networks because they overfitted the noise in the dataset and the predictions made by the model were too random, leading to a decrease in reliability for detecting the exact timing of anomalies and consequently a rise in false positives.

We analysed the time series of the different features that were found to be significant for each type of attack, which are those with the highest values for the partial derivatives. In particular, we examined the time series of the variables *swin*, *buffer_overflow* and *dollar_payload*. When analysing these time series, we can only obtain the exact time of one of the cyberattacks, as neither of them is representative of both attacks. However, by examining the time series of the number of bytes sent, we can detect the timing of both attacks. As mentioned in Section 3.3.6, we performed time series analysis for the most relevant source and destination IP address pairs in the final anomaly lists, in our case 10.0.2.10 and 10.0.2.5, and 10.0.2.10 and 10.0.2.7.

Below we detail the exact statistical analysis for the source and destination IP addresses 10.0.2.10 and 10.0.2.5 respectively, which is the most repeated combination of source and destination addresses in the final anomaly detection results for both

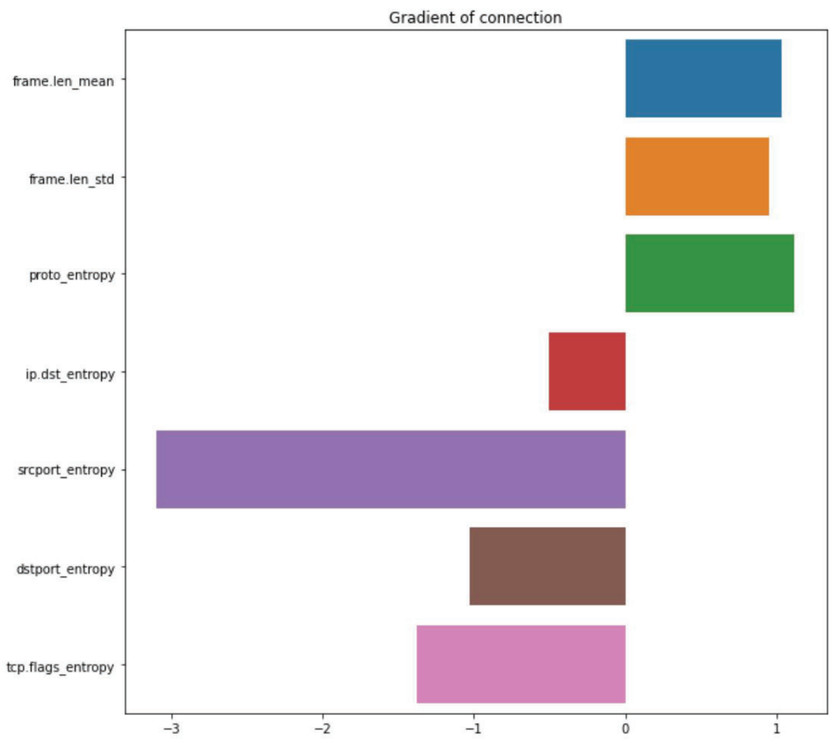


Figure 4.8: Explanations for the *time_window* pre-processing based on partial derivatives for a data point belonging to the lateral movement attack with ID: 10.0.2.10, 14:52.

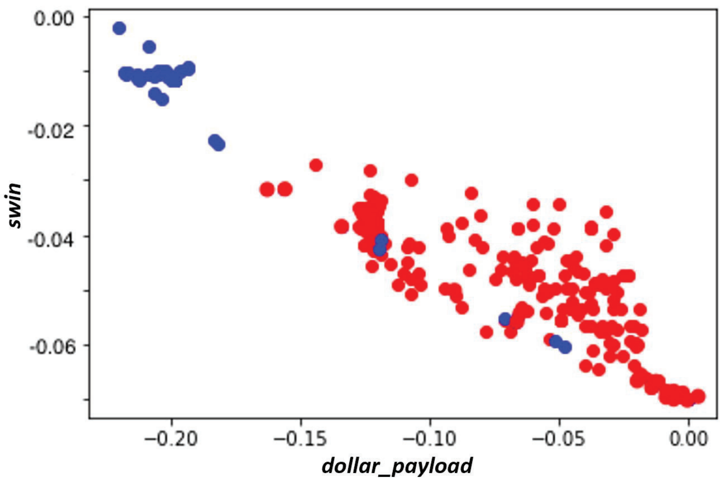


Figure 4.9: Distribution of anomalous points of the port scanning (red) and lateral movement (blue) attacks with respect to the variables *dollar_payload* (X-axis) and *swin* (Y-axis).

models.

First, we decompose the selected time series into trend, seasonal and residual, in order to get rid of the noise present in traffic data. Using the seasonal series for *normal* and *attack* data of source IP address 10.0.2.5, we could correlate the time series after adjusting the "lag" between them, as illustrated in Figure 4.12. We can observe

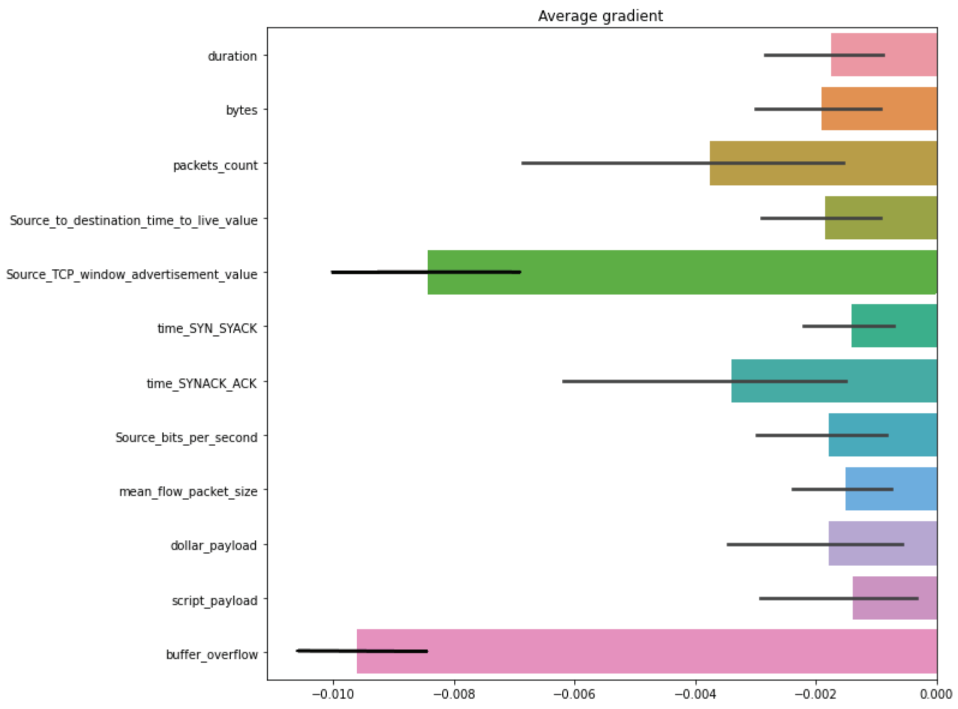


Figure 4.10: Average of the normalized partial derivatives for the port scanning attack cluster.

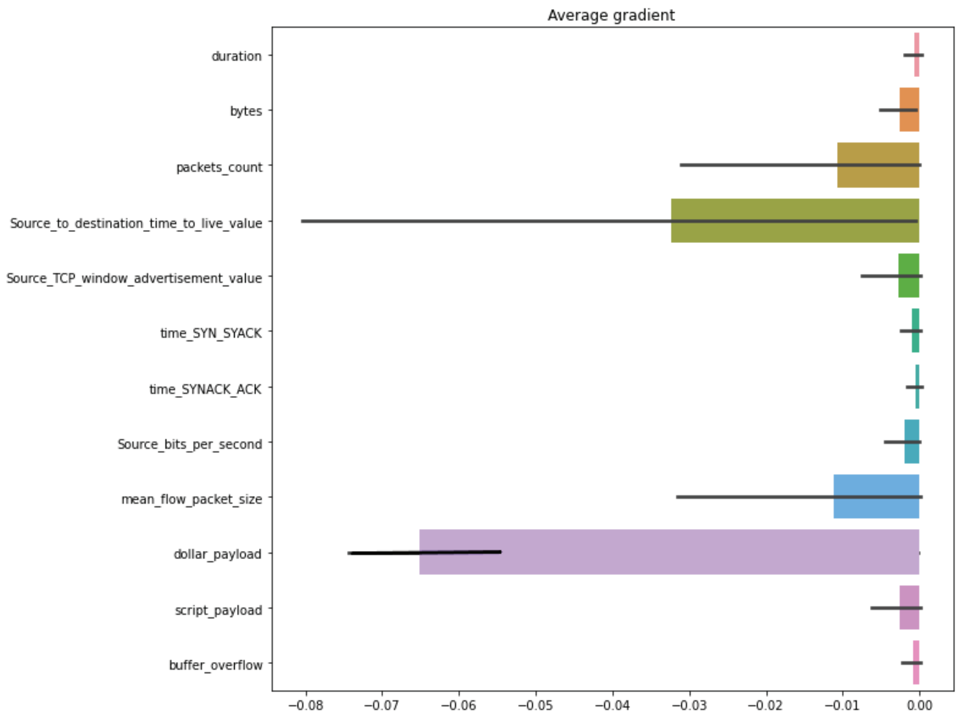


Figure 4.11: Average of the normalized partial derivatives for the lateral movement attack cluster.

that there is a clear periodicity in the normal behaviour data, that also appears in the attack data but which breaks down at the moments when an attack is perpetrated. We then studied Pearson correlation between the adjusted series, observing a clear difference when both complete series are analysed (0.67) versus when we limit ourselves to the time interval between the two attacks (0.91). Also, points where the difference between both series is remarkable were marked as suspicious (for clarity they are not shown in the figure) and stored in a vector.

In addition, we examined the time series of the attacker's and victim's IP addresses, as shown in Figure 4.13, and performed rolling window correlation synchrony between them, looking for local maxima. We finally correlated the local maxima points with the suspicious points of the previous step to obtain the exact time of attacks, which are 13:39 for the port scanning attack and 14:52 for the lateral movement.

It is important to note that although the time series analysis could give us a more accurate value of the time of the attack, the accuracy of the 2-minute window obtained in the final merged tables for both models is sufficient in an industrial environment, which is our target.

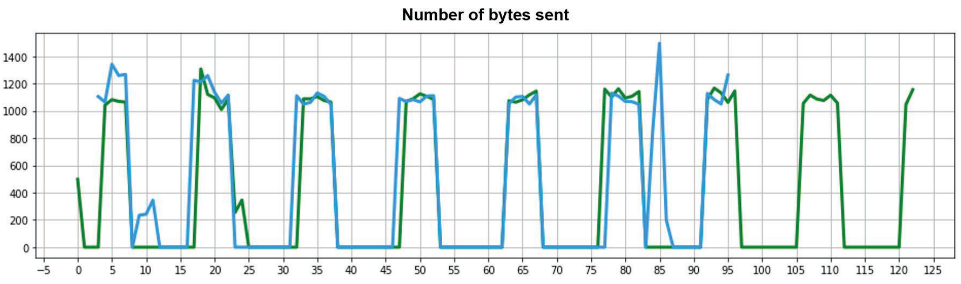


Figure 4.12: Time series correlation of *normal* (green) and *attack* (blue) data for destination IP address 10.0.2.5.

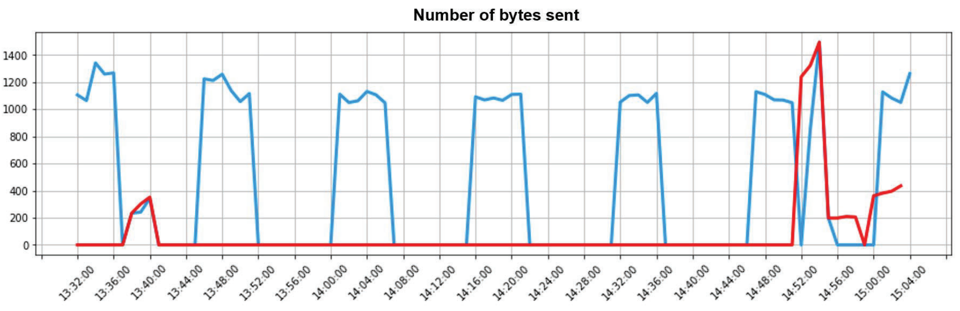


Figure 4.13: Attacker 10.0.2.10 (red) and victim 10.0.2.5 (blue) time series for the *attack* dataset.

Chapter 5

Conclusions and future research

5.1 Conclusions

In this work, we have presented an anomaly-based network IDS using semi-supervised models. Specifically, we have compared the performance of two models: a semi-parametric Bayesian network (and their hybrid version when needed) and a variational autoencoder. Although we have designed our system for the field of cybersecurity, it can be applied to any other field for anomaly detection due to its adaptability to the type of input data.

The proposed system has been tested on network traffic data provided by *Titanium Industrial Security S.L.* There, we have mainly dealt with two different types of cyber-attacks: a port scanning attack along with a lateral movement attack on two different devices of the simulated network. While the first is easier to detect, as a single machine is scanning all the ports of all available devices on the network; the second is more challenging due to its stealthy nature, where the attacker uses evasive techniques for trying not to raise any alarm.

Due to the fact that the attack dataset is not properly labelled, the evaluation stage has been a difficult task and may not have been as accurate and precise as it could have been.

It is also important to highlight that, unlike others, our system makes no prior assumptions of the underlying distribution of continuous variables. In many papers, a Gaussian distribution is assumed, but by performing statistical tests we were able to verify that this is not always the case.

In scenarios where the pre-processed dataset, which is the input to our semi-supervised model, contains both continuous and discrete variables, the use of a HSPBN is an excellent choice, because it can perfectly fits the input data, that is, it can model both categorical and continuous features, without prior assumptions about the probability distribution of the latter. Nevertheless, the VAE we have employed cannot model discrete probability distributions, thus we had to treat discrete variables as continuous, which leads to a slight deterioration of its performance. It would be interesting to investigate this last point further.

In order to create a system that addresses one of the weaknesses of IDSs, which is the detection of a high number of false alarms, we decided to correlate the anomaly detection results using two alternative pre-processing of the raw traffic data. This results in a more robust system.

Finally, special attention has been paid to the explainability aspects of our system. Given that from the outset, our aim was not only to have an efficient tool for anomaly detection, but also an explainable one, individual and collective explanations are therefore a key element in our work.

For this purpose, we have tried to identify the features that are most relevant to the model to mark an event as an anomaly. To achieve this, we have analysed the partial derivate for each variable of a flagged anomalous event with respect to the objective function of the used model (SPBN or VAE).

In addition, we grouped anomalies with similar information for their partial derivatives into clusters, in order to gather events belonging to the same type of cyberattack. The collective information or fingerprint of each type of cyberattack, which is obtained from the average of the individual partial derivates of the events that are part of the same cluster, can be seen as a summary of collective explanations. This is very convenient and helpful for a cybersecurity analyst, who may have to examine malicious events afterwards.

The anomaly detection results obtained for the attack dataset were remarkable for both models, capturing both types of cyberattacks and identifying the attacker's IP address. Besides, the cybersecurity expert reviewed the results and explanations of our system, and admitted that the individual and collective information to explain the malicious events was very useful and overall corresponded to what he expected for each particular type of cyberattack.

5.2 Future work

Despite the overall good performance of our system, some refinements or improvements could be made. Hence there are several research lines that can be further investigated in the future:

- An important aspect that could be studied further is scalability, as we have been working with very limited datasets, not in terms of number of devices, as we mentioned that the proposed system will be implemented locally in the network, but with regards to the number of events or the duration in time.

  Although we have taken this into account when developing the system, it would be interesting to test its performance with larger datasets. When doing so, several decisions would have to be made. For example, we would have to decide how often the system will output a list of anomalous events along with their corresponding explanations.

  Moreover, in this case where we are showing anomalies to the system user with a certain periodicity, the clustering stage must necessarily be performed using data stream clustering techniques such as the one we described in section 3.3.5 with the adaptive k-means algorithm. As before, anomalies which form part of the same type of cyberattack will usually come in a sequence, which serves as an advantage to decide whether new anomalies belong to an existing cluster or otherwise a new cluster should be formed.

Unlike the case of *Titanium*'s dataset, in a general situation it is not possible to store the whole explanation data, so it is necessary to use a special data structure to maintain a synthesis of them. Storing an agglomerative sum or storing only representative samples of the data are two popular alternative structures. Moreover, users are often interested in the most recent data instances rather than in the previous ones. This situation creates a requirement of obsolescence for previous data instances. In data stream clustering, it is solved by time window models, such as damped window, landmark window or sliding window models.

Finally, it is important to note that the adaptive k-means algorithm has a limitation because of being k-means based, which is that only hyper-spherical clusters can be detected. It would be interesting to further try other data stream clustering algorithms, such as I-HASTREAM [Hassani *et al.*, 2016] that claims to have no limitations in this sense.

- Although our system will run locally, as mentioned above, it is important to carry out future work on how the different local systems will comunicate with each other to obtain a centralised response.
- Another essential element is adaptability to new network environments. By this we do not mean deploying the system in a new client company, because when this is done you first have to train the normal behaviour model with the corresponding data, which is straighforward due to the design of our system; but we do mean once the system is up and running, normal behaviour of the network may change and we have to cope with this.

  It is true that in industrial environments, on which we focus, these changes in normal behaviour are gradual and not sudden as might be the case in IT environments. However, it is necessary that the system can self-adapt to this new normality, otherwise there could be an increase in false positives, which deteriorates the reliability of the system.

  Then, investigating gradual concept drift might be more effective than constantly retraining the model.
- It would be interesting to analyse and include new variables in the pre-processed datasets which could provide deeper insights to detect malicious instances. This should be done together with the help of a cybersecurity expert who could asses which features would be meaningful and useful in generating explanations.

  A possible example could be for the *time_window* pre-processing, the percentage of particular sequences of TCP flags in the time window. Particular sequences would have to be chosen with the help of a cybersecurity expert, selecting those that would be useful in case a security analyst had to manually revise the output of our model.

  As we want a lightweight system, we must carefully choose the variables that we include so as not to degrade its efficiency and speed. Moreover, we have tried to ensure that our system does not focus on a particular type of cyberattack, but rather that it is multidisciplinary in the sense of detecting attacks of different natures. Hence, new variables should not be very specific of a particular type of cyberattack.

- As mentioned above, to best fit categorical data, a VAE model that can handle discrete probability distributions should be analysed.

  It would also be interesting to study other variants of VAE model such as the conditional VAE (CVAE) model [Sohn *et al.*, 2015]. The CVAE was developed as an extension of the VAE to allow for additional auxiliary labels that are available. Hence, with CVAE, we can consider training the anomaly detection system from multiple data sources that have different behaviours, as the normal behaviour of a particular device might differ greatly from others.

- To refine the decisions of our system and improve the accuracy of future predictions, feedback from the cybersecurity analyst could be incorporated. Two possibilities for this are to either add a penalty to the anomaly score of events with similar characteristics to the ones the analyst has discarded, or train with the provided feedback a supervised model in conjunction with semi-supervised models to predict attacks. As more feedback is gathered, the model is constantly refined.

- Another research direction would be to adjust the system to be able to perform real-time anomaly detection together with the generation of explanations. This would be very useful in reducing the time lag from detection of the cyberattack until action is taken.

  With regards to the clustering stage, up to the recent years, most of data stream clustering algorithms were online-offline algorithms. A synopsis of the data is employed in the online phase (synopsis of the data stream is updated when a new instance is received, thus remaining up-to-date) and the final clusters are generated in the offline phase. In this type of algorithms, the offline phase is executed periodically or upon user request. Therefore, final clustering results are obtained with a latency and they are not up to date most of the times.

  However, there are several recent fully online algorithms in the literature, thus we could achieve real-time clustering of anomaly explanations. Fully online algorithms maintain the final clustering results up to date. Therefore, users get the results with no latency. An example of a real-time clustering algorithm we could use is online k-means [Cohen-Addad *et al.*, 2021].

- In terms of the determination of thresholds before correlating the lists of results from both pre-processing, we could investigate different alternatives to refine our choice for each specific industrial environment.

- Finally, it would be interesting to consider a hybrid approach for IDS, that is combining our anomaly-based system with a signature-based IDS that could lead to improved detection accuracy. Or even, develop a host-based IDS in combination with our system, which could also exploit event log information.

Bibliography

Abbes, T., Bouhoula, A., & Rusinowitch, M. (2010). Efficient decision tree for protocol analysis in intrusion detection. *International Journal of Security and Networks*, *5*(4), 220–235. https://doi.org/10.1504/IJSN.2010.037661

Agarap, A. F. M. (2018). A neural network architecture combining gated recurrent unit (GRU) and support vector machine (SVM) for intrusion detection in network traffic data. *Proceedings of the 2018 10th International Conference on Machine Learning and Computing*, 26–30.

Aggarwal, C. C. (2017). *Outlier Analysis*. Springer.

Aldweesh, A., Derhab, A., & Emam, A. (2020). Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues. *Knowledge-Based Systems*, *189*, 105–124.

An, X., Jutla, D., & Cercone, N. (2006). Privacy intrusion detection using dynamic Bayesian networks. *Proceedings of the 8th International Conference on Electronic Commerce*, 208–215. https://doi.org/10.1145/1151454.1151493

Anderson, J. P. (1980). Computer security threat monitoring and surveillance. *Technical Report, James P. Anderson Company*.

Ariyaluran Habeeb, R. A., Nasaruddin, F., Gani, A., Amanullah, M. A., Hashem, I., Ahmed, E., & Imran, M. (2019). Clustering-based real-time anomaly detection—a breakthrough in big data technologies. *Transactions on Emerging Telecommunications Technologies*, e3647. https://doi.org/10.1002/ett.3647

Arthur, D., & Vassilvitskii, S. (2007). K-means++: The advantages of careful seeding. *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1027–1035.

Atienza, D. (2021). *Nonparametric Models and Bayesian Networks. Applications to Anomaly Detection* (Doctoral dissertation). ETSI Informáticos, Universidad Politécnica de Madrid.

Atienza, D., Larrañaga, P., & Bielza, C. (2022a). Hybrid semiparametric Bayesian networks. *TEST*, *31*, 299–327. https://doi.org/10.1007/s11749-022-00812-3

Atienza, D., Larrañaga, P., & Bielza, C. (2022b). PyBNesian: An etensible Python package for Bayesian networks. *Neurocomputing, accepted for publication.*

Atienza, D., Larrañaga, P., & Bielza, C. (2022c). Semiparametric Bayesian networks. *Information Sciences*, *584*, 564–582. https://doi.org/https://doi.org/10.1016/j.ins.2021.10.074

Ayyarao, T. S., & Kiran, I. R. (2021). A two-stage Kalman filter for cyber-attack detection in automatic generation control system. *Journal of Modern Power Systems and Clean Energy*, (1), 50–59.

Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. (2020). Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, *58*, 82–115. https://doi.org/https://doi.org/10.1016/j.inffus.2019.12.012

Barrett, B., Camuto, A., Willetts, M., & Rainforth, T. (2022). Certifiably robust variational autoencoders. *International Conference on Artificial Intelligence and Statistics*, 3663–3683.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. *Neural Networks: Tricks of the Trade*, 437–478.

Blei, D. M., Kucukelbir, A., & McAuliffe, J. D. (2016). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, *112*, 859–877.

Boukabour, S., & Masmoudi, A. (2021). Semiparametric Bayesian networks for continuous data. *Communications in Statistics - Theory and Methods*, *50*(24), 5974–5996. https://doi.org/10.1080/03610926.2020.1738486

Boukerche, A., Zheng, L., & Alfandi, O. (2020). Outlier detection: Methods, models, and classification. *ACM Computing Surveys*, *53*(3). https://doi.org/10.1145/3381028

Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). LOF: Identifying density-based local outliers. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 93–104. https://doi.org/10.1145/342009.335388

Camuto, A., Willetts, M., Roberts, S., Holmes, C., & Rainforth, T. (2021). Towards a theoretical understanding of the robustness of variational autoencoders. *International Conference on Artificial Intelligence and Statistics*, 3565–3573.

Carvalho, D. V., Pereira, E. M., & Cardoso, J. S. (2019). Machine learning interpretability: A survey on methods and metrics. *Electronics*, *8*(8). https://doi.org/10.3390/electronics8080832

Caudle, K., Karlsson, C., & Pyeatt, L. D. (2015). Using density estimation to detect computer intrusions. *Proceedings of the 2015 ACM International Workshop on International Workshop on Security and Privacy Analytics*, 43–48. https://doi.org/10.1145/2713579.2713584

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, *41*(3). https://doi.org/10.1145/1541880.1541882

Chawla, N. V. (2009). Data mining for imbalanced datasets: An overview. *Data Mining and Knowledge Discovery Handbook*, 875–886.

Chen, J., Su, C., Yeh, K.-H., & Yung, M. (2018). Special issue on advanced persistent threat. *Future Generation Computer Systems*, *79*, 243–246.

Cohen-Addad, V., Guedj, B., Kanade, V., & Rom, G. (2021). Online k-means clustering. *International Conference on Artificial Intelligence and Statistics*, 1126–1134.

Day, J. D., & Zimmermann, H. (1983). The OSI reference model. *Proceedings of the IEEE*, *71*(12), 1334–1340.

Derks, I. P., & de Waal, A. (2020). A taxonomy of explainable Bayesian networks. *Artificial Intelligence Research*, 220–235.

Dhanabal, L., & Shantharajah, S. P. (2015). A study on NSL-KDD dataset for intrusion detection system based on classification algorithms. *Computer Science*, *4*(6), 446–452.

Djenouri, Y., Belhadi, A., Lin, J. C.-W., Djenouri, D., & Cano, A. (2019). A survey on urban traffic anomalies detection algorithms. *IEEE Access*, *7*, 12192–12205. https://doi.org/10.1109/ACCESS.2019.2893124

Dor, D., & Tarsi, M. (1992). A simple algorithm to construct a consistent extension of a partially oriented graph. *Technical report R-185, Cognitive Systems Laboratory, Department of Computer Science, UCLA.*

Ennaji, S., Akkad, N. E., & Haddouch, K. (2021). A powerful ensemble learning approach for improving network intrusion detection system (NIDS). *2021 5th International Conference On Intelligent Computing in Data Sciences*, 1–6. https://doi.org/10.1109/ICDS53782.2021.9626727

Ertoz, L., Eilertson, E., Lazarevic, A., Tan, P.-N., Kumar, V., Srivastava, J., & Dokas, P. (2004). MINDS-Minnesota intrusion detection system. *Next Generation Data Mining*, 199–218.

Eskin, E. (2000). Anomaly detection over noisy data using learned probability distributions. *Proceedings of the 17th International Conference on Machine Learning*, 255–262.

Evangelou, M., & Adams, N. M. (2020). An anomaly detection framework for cybersecurity data. *Computers & Security*, *97*, 101941. https://doi.org/https://doi.org/10.1016/j.cose.2020.101941

Falcão, F., Zoppi, T., Silva, C. B. V., Santos, A., Fonseca, B., Ceccarelli, A., & Bondavalli, A. (2019). Quantitative comparison of unsupervised anomaly detection algorithms for intrusion detection. *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 318–327.

Fan, W., Miller, M., Stolfo, S., Lee, W., & Chan, P. (2004). Using artificial anomalies to detect unknown and known network intrusions. *Knowledge and Information Systems*, *6*(5), 507–527.

Fei, L., Ting, K. M., & Zhou, Z.-H. (2008). Isolation forest. *2008 8th IEEE International Conference on Data Mining*, 413–422. https://doi.org/10.1109/ICDM.2008.17

Fernando, T., Gammulle, H., Denman, S., Sridharan, S., & Fookes, C. (2021). Deep learning for medical anomaly detection–a survey. *ACM Computing Surveys*, *54*(7), 1–37.

Fiore, U., Palmieri, F., Castiglione, A., & De Santis, A. (2013). Network anomaly detection with the restricted Boltzmann machine. *Neurocomputing*, *122*, 13–23. https://doi.org/https://doi.org/10.1016/j.neucom.2012.11.050

Forgy, E. W. (1965). Cluster analysis of multivariate data : Efficiency versus interpretability of classifications. *Biometrics*, *21*, 768–769.

Gao, X., Shan, C., Hu, C., Niu, Z., & Liu, Z. (2019). An adaptive ensemble machine learning model for intrusion detection. *IEEE Access*, *7*, 82512–82521.

García, N. M. (2019). Multi-agent system for anomaly detection in industry 4.0 using machine learning techniques. *Advances in Distributed Computing and Artificial Intelligence Journal*, *8*(4), 33–40.

García-Teodoro, P., Díaz-Verdejo, J., Maciá-Fernández, G., & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, *28*(1), 18–28. https://doi.org/https://doi.org/10.1016/j.cose.2008.08.003

Geiger, D., & Heckerman, D. (1994). Learning Gaussian networks. *Proceedings of the 10th International Conference on Uncertainty in Artificial Intelligence*, 235–243.

Geiger, D., & Heckerman, D. (2002). Parameter priors for directed acyclic graphical models and the characterization of several probability distributions. *The Annals of Statistics*, *30*(5), 1412–1440. https://doi.org/10.1214/aos/1035844981

Geiger, D., Verma, T., & Pearl, J. (1990). D-separation: From theorems to algorithms. *Uncertainty in Artificial Intelligence* (pp. 139–148). North-Holland. https://doi.org/https://doi.org/10.1016/B978-0-444-88738-2.50018-X

Ghafouri, A., Abbas, W., Laszka, A., Vorobeychik, Y., & Koutsoukos, X. (2016). Optimal thresholds for anomaly-based intrusion detection in dynamical environments. *International Conference on Decision and Game Theory for Security*, 415–434.

Glover, F., & Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers.

Goldstein, M., & Dengel, A. R. (2012). Histogram-based outlier score (HBOS): A fast unsupervised anomaly detection algorithm. *In Proceedings of 35th German Conference on Artificial Intelligence*, 59–63.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, 2672–2680.

Greggio, N. (2013). Learning anomalies in IDSs by means of multivariate finite mixture models. *2013 IEEE 27th International Conference on Advanced Information Networking and Applications*, 251–258. https://doi.org/10.1109/AINA.2013.151

Hannan, A., Gruhl, C., & Sick, B. (2021). Anomaly based resilient network intrusion detection using inferential autoencoders. *2021 IEEE International Conference on Cyber Security and Resilience*, 1–7. https://doi.org/10.1109/CSR51186.2021.9527980

Hassani, M., Spaus, P., Cuzzocrea, A., & Seidl, T. (2016). I-HASTREAM: Density-based hierarchical clustering of big data streams and its application to big graph analytics tools. *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 656–665. https://doi.org/10.1109/CCGrid.2016.102

Hofmann, R., & Tresp, V. (1995). Discovering structure in continuous variables using Bayesian networks. *Proceedings of the 8th International Conference on Neural Information Processing Systems*, 500–506.

Holder, E., & Wang, N. (2021). Explainable artificial intelligence (XAI) interactively working with humans as a junior cyber analyst. *Human-Intelligent Systems Integration*, *3*(2), 139–153. https://doi.org/10.1007/s42454-020-00021-z

Hu, W., Gao, J., Li, B., Wu, O., Du, J., & Maybank, S. (2020). Anomaly detection using local kernel density estimation and context-based regression. *IEEE Transactions on Knowledge and Data Engineering*, *32*(2), 218–233. https://doi.org/10.1109/TKDE.2018.2882404

Iliyasu, A. S., & Deng, H. (2022). N-GAN: a novel anomaly-based network intrusion detection with generative adversarial networks. *International Journal of Information Technology*, *3*, 1–11.

Imrana, Y., Xiang, Y., Ali, L., & Abdul-Rauf, Z. (2021). A bidirectional LSTM deep learning approach for intrusion detection. *Expert Systems with Applications*, *185*, 115524. https://doi.org/https://doi.org/10.1016/j.eswa.2021.115524

Jabbar, M., Aluvalu, R., & Reddy S, S. S. (2017). RFAODE: A novel ensemble intrusion detection system. *Procedia Computer Science*, *115*, 226–234. https://doi.org/https://doi.org/10.1016/j.procs.2017.09.129

Jain, M., & Kaur, G. (2021). Distributed anomaly detection using concept drift detection based hybrid ensemble techniques in streamed network data. *Cluster Computing*, *24*(3), 2099–2114.

Jan, S. U., Ahmed, S., Shakhov, V., & Koo, I. (2019). Toward a lightweight intrusion detection system for the internet of things. *IEEE Access*, *7*, 42450–42471. https://doi.org/10.1109/ACCESS.2019.2907965

Jing, S., Li, M., Sun, Y., & Zhang, Y. (2021). Research on prediction of attack behavior based on HMM. *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, *4*, 1580–1583.

Jinwon, A., & Sungzoon, C. (2015). Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, *2*, 1–18.

Kamalov, F. (2020). Kernel density estimation based sampling for imbalanced class distribution. *Information Sciences*, *512*, 1192–1201.

Kang, H., Liu, B., Mišić, J., Mišić, V. B., & Chang, X. (2020). Assessing security and dependability of a network system susceptible to lateral movement attacks. *2020 International Conference on Computing, Networking and Communications*, 513–517.

Khraisat, A., Gondal, I., Vamplew, P., & Kamruzzaman, J. (2019). Survey of intrusion detection systems: Techniques, datasets and challenges. *Cybersecurity*, *2*(1), 1–22.

Kind, A., Stoecklin, M. P., & Dimitropoulos, X. A. (2009). Histogram-based traffic anomaly detection. *IEEE Transactions on Network and Service Management*, *6*, 110–121.

Kingma, D., & Welling, M. (2014). Auto-encoding variational Bayes. *Proceedings of the 2nd International Conference on Learning Representations*, 1–14.

Koc, L., Mazzuchi, T. A., & Sarkani, S. (2012). A network intrusion detection system based on a hidden naïve Bayes multiclass classifier. *Expert Systems with Applications*, *39*(18), 13492–13500. https://doi.org/https://doi.org/10.1016/j.eswa.2012.07.009

Koller, D., & Friedman, N. (2009). *Probabilistic Graphical Models. Principles and Techniques*. The MIT Press. https://mitpress.mit.edu/books/probabilistic-graphical-models

Kulkarni, A., & Bush, S. (2006). Detecting distributed denial-of-service attacks using Kolmogorov complexity metrics. *Journal of Network and Systems Management*, *14*(1), 69–80.

Laird, N. (1993). 14 The EM algorithm. *Computational Statistics* (pp. 509–520). Elsevier. https://doi.org/https://doi.org/10.1016/S0169-7161(05)80138-5

Leevy, J. L., & Khoshgoftaar, T. M. (2020). A survey and analysis of intrusion detection models based on CSE-CIC-IDS2018 Big Data. *Journal of Big Data*, *7*(1), 104. https://doi.org/10.1186/s40537-020-00382-x

Leslie, D. (2019). Understanding artificial intelligence ethics and safety: A guide for the responsible design and implementation of AI systems in the public sector. *The Alan Turing Institute*.

Li, L., Zhang, H., Peng, H., & Yang, Y. (2018). Nearest neighbors based density peaks approach to intrusion detection. *Chaos, Solitons & Fractals*, *110*, 33–40. https://doi.org/https://doi.org/10.1016/j.chaos.2018.03.010

Lipton, Z. C. (2018). The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, *16*(3), 31–57. https://doi.org/10.1145/3236386.3241340

Liu, H., & Lang, B. (2019). Machine learning and deep learning methods for intrusion detection systems: A survey. *Applied Sciences*, *9*(20), 4396.

Lomio, F., Baselga, D. M., Moreschini, S., Huttunen, H., & Taibi, D. (2020). RARE: a labeled dataset for cloud-native memory anomalies. *Proceedings of the 4th ACM SIGSOFT International Workshop on Machine-Learning Techniques for Software-Quality Evaluation*, 19–24.

Lopez Pinaya, W. H., Vieira, S., Garcia-Dias, R., & Mechelli, A. (2020). Chapter 11 - Autoencoders. *Machine learning* (pp. 193–208). Academic Press. https://doi.org/https://doi.org/10.1016/B978-0-12-815739-8.00011-0

Ludwig, S. A. (2017). Intrusion detection of multiple attack classes using a deep neural net ensemble. *2017 IEEE Symposium Series on Computational Intelligence*, 1–7. https://doi.org/10.1109/SSCI.2017.8280825

Ludwig, S. A. (2019). Applying a neural network ensemble to intrusion detection. *Journal of Artificial Intelligence and Soft Computing Research*, *9*, 177–188.

Mahdavifar, S., & Ghorbani, A. A. (2020). DeNNeS: Deep embedded neural network expert system for detecting cyber attacks. *Neural Computing and Applications*, *32*(18), 14753–14780. https://doi.org/10.1007/s00521-020-04830-w

Malik, A. J., & Khan, F. A. (2018). A hybrid technique using binary particle swarm optimization and decision tree pruning for network intrusion detection. *Cluster Computing*, *21*(1), 667–680. https://doi.org/10.1007/s10586-017-0971-8

Martí, L., Sanchez-Pi, N., Molina, J. M., & Garcia, A. C. B. (2015). Anomaly detection based on sensor data in petroleum industry applications. *Sensors*, *15*(2), 2774–2797.

Mehmood, A., Mukherjee, M., Ahmed, S. H., Song, H., & Malik, K. M. (2018). NBC-MAIDS: Naïve Bayesian classification technique in multi-agent system-enriched IDS for securing IoT against DDoS attacks. *The Journal of Supercomputing*, *74*(10), 5156–5170. https://doi.org/10.1007/s11227-018-2413-7

Meng, Y., & Kwok, L.-f. (2012). Adaptive false alarm filter using machine learning in intrusion detection. *Practical Applications of Intelligent Systems*, 573–584.

Mihaljević, B., Bielza, C., & Larrañaga, P. (2021). Bayesian networks for interpretable machine learning and optimization. *Neurocomputing*, *456*(100), 648–665. https://doi.org/10.1016/j.neucom.2021.01.138

Moustafa, N., Creech, G., & Slay, J. (2017). Big data analytics for intrusion detection system: Statistical decision-making using finite Dirichlet mixture models. *Data Analytics and Decision Support for Cybersecurity: Trends, Methodologies and Applications* (pp. 127–156). Springer International Publishing. https://doi.org/10.1007/978-3-319-59439-2_5

Moustafa, N., & Slay, J. (2015). UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). *2015 Military Communications and Information Systems Conference*, 1–6. https://doi.org/10.1109/MilCIS.2015.7348942

Noble, C. C., & Cook, D. J. (2003). Graph-based anomaly detection. *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 631–636. https://doi.org/10.1145/956750.956831

Ogbechie, A., Díaz-Rozo, J., Larrañaga, P., & Bielza, C. (2017). Dynamic Bayesian network-based anomaly detection for in-process visual inspection of laser surface heat treatment. *Machine Learning for Cyber Physical Systems* (pp. 17–24). CRC Press.

Pang, G., Shen, C., Cao, L., & Hengel, A. V. D. (2021). Deep learning for anomaly detection: A review. *ACM Computing Surveys*, *54*(2), 1–38. https://doi.org/10.1145/3439950

Parzen, E. (1962). On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, *33*(3), 1065–1076. https://doi.org/10.1214/aoms/1177704472

Paula, E. L., Ladeira, M., Carvalho, R. N., & Marzagao, T. (2016). Deep learning anomaly detection as support fraud investigation in Brazilian exports and anti-money laundering. *2016 15th IEEE International Conference on Machine Learning and Applications*, 954–960.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann. https://doi.org/10.1016/c2009-0-27609-4

Peng, W., Li, Y., Zhang, Z., Hu, T., Li, Z., & Liu, D. (2019). An optimization method for intrusion detection classification model based on deep belief network. *IEEE Access*, *7*, 87593–87605. https://doi.org/10.1109/ACCESS.2019.2925828

Provost, F. (2000). Machine learning from imbalanced data sets. *Proceedings of the AAAI'2000 Workshop on Imbalanced Datasets*, *68*(2000), 1–3.

Puschmann, D., Barnaghi, P., & Tafazolli, R. (2017). Adaptive clustering for dynamic IoT data streams. *IEEE Internet of Things Journal*, *4*(1), 64–74. https://doi.org/10.1109/JIOT.2016.2618909

Rosenblatt, M. (1956). Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, *27*(3), 832–837. https://doi.org/10.1214/aoms/1177728190

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, *1*(5), 206–215. https://doi.org/10.1038/s42256-019-0048-x

Sakr, M. M., Tawfeeq, M. A., & El-Sisi, A. B. (2019). Network intrusion detection system based PSO-SVM for cloud computing. *International Journal of Computer Network and Information Security*, *10*(3), 22.

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, *6*(2), 461–464. https://doi.org/10.1214/aos/1176344136

Sedik, A., Emara, H. M., Hamad, A., Shahin, E. M., A El-Hag, N., Khalil, A., Ibrahim, F., Elsherbeny, Z. M., Elreefy, M., Zahran, O., *et al.* (2019). Efficient anomaly detection from medical signals and images. *International Journal of Speech Technology*, *22*(3), 739–767.

Shachter, R. D., & Kenley, C. R. (1989). Gaussian influence diagrams. *Management Sciences*, *35*(5), 527–550.

Shen, X., & Agrawal, S. (2006). Kernel density estimation for an anomaly based intrusion detection system. *Proceedings of the 2006 International Conference on Machine Learning; Models, Technologies & Applications*, 161–167.

Shone, N., Ngoc, T. N., Phai, V. D., & Shi, Q. (2018). A deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *2*(1), 41–50. https://doi.org/10.1109/TETCI.2017.2772792

Singh, N. B., Singh, M. M., Sarkar, A., & Mandal, J. K. (2021). A novel wide & deep transfer learning stacked GRU framework for network intrusion detection. *Journal of Information Security and Applications*, *61*, 102899.

Smiti, A. (2020). A critical overview of outlier detection methods. *Computer Science Review*, *38*, 100306. https://doi.org/https://doi.org/10.1016/j.cosrev.2020.100306

Sohn, K., Yan, X., & Lee, H. (2015). Learning structured output representation using deep conditional generative models. *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, 3483–3491.

Spirtes, P. L., Glymour, C., & Scheines, R. (2001). *Causation, Prediction, and Search, 2nd Edition*. The MIT Press.

Strobl, E. V., Zhang, K., & Visweswaran, S. (2018). Approximate kernel-based conditional independence tests for fast non-parametric causal discovery. *Journal of Causal Inference*, *7*(1), 10–34.

Sun, X., Dai, J., Liu, P., Singhal, A., & Yen, J. (2018). Using Bayesian networks for probabilistic identification of zero-day attack paths. *IEEE Transactions on Information Forensics and Security*, *13*(10), 2506–2521. https://doi.org/10.1109/TIFS.2018.2821095

Szczepański, M., Choraś, M., Pawlicki, M., & Kozik, R. (2020). Achieving explainability of intrusion detection system by hybrid oracle-explainer approach. *2020 International Joint Conference on Neural Networks*, 1–8. https://doi.org/10.1109/IJCNN48605.2020.9207199

Tang, T. A., Mhamdi, L., McLernon, D., Zaidi, S. A. R., & Ghogho, M. (2016). Deep learning approach for network intrusion detection in software defined networking. *2016 International Conference on Wireless Networks and Mobile Communications*, 258–263. https://doi.org/10.1109/WINCOM.2016.7777224

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. (2009). A detailed analysis of the KDD CUP 99 data set. *IEEE Symposium. Computational Intelligence for Security and Defense Applications, CISDA*, *2*, 1–6. https://doi.org/10.1109/CISDA.2009.5356528

Teng, S., Wu, N., Zhu, H., Teng, L., & Zhang, W. (2018). SVM-DT-based adaptive and collaborative intrusion detection. *IEEE/CAA Journal of Automatica Sinica*, *5*(1), 108–118. https://doi.org/10.1109/JAS.2017.7510730

Thiprungsri, S., & Vasarhelyi, M. A. (2011). Cluster analysis for anomaly detection in accounting data: An audit approach. *The International Journal of Digital Accounting Research*, *11*, 69–84.

Tian, L., & Jianwen, W. (2009). Research on network intrusion detection system based on improved k-means clustering algorithm. *2009 International Forum on Computer Science-Technology and Applications*, *1*, 76–79. https://doi.org/10.1109/IFCSTA.2009.25

Tsamardinos, I., Brown, L. E., & Aliferis, C. F. (2006). The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, *65*, 31–78.

Turcotte, M., Kent, A., & Hash, C. (2018). Data Science for Cyber-Security. *Security Science and Technology* (pp. 1–22). World Scientific.

Vanhoeyveld, J., Martens, D., & Peeters, B. (2020). Value-added tax fraud detection with scalable anomaly detection techniques. *Applied Soft Computing*, *86*, 105895.

Vercruyssen, V., Meert, W., Verbruggen, G., Maes, K., Bäumer, R., & Davis, J. (2018). Semi-supervised anomaly detection with an application to water analytics. *2018 IEEE International Conference on Data Mining*, 527–536. https://doi.org/10.1109/ICDM.2018.00068

Viegas, E. K., Santin, A. O., & Oliveira, L. S. (2017). Toward a reliable anomaly-based intrusion detection in real-world environments. *Computer Networks*, *127*, 200–216.

Vigano, L., & Magazzeni, D. (2018). Explainable security. *IJCAI/ECAI 2018 Workshop on Explainable Artificial Intelligence (XAI)* (pp. 293–300).

Villa-Pérez, M. E., Álvarez-Carmona, M. Á., Loyola-González, O., Medina-Pérez, M. A., Velazco-Rossell, J. C., & Choo, K.-K. R. (2021). Semi-supervised anomaly detection algorithms: A comparative summary and future research directions. *Knowledge-Based Systems*, *218*, 106878. https://doi.org/https://doi.org/10.1016/j.knosys.2021.106878

Wang, Z., Zeng, Y., Liu, Y., & Li, D. (2021). Deep belief network integrating improved kernel-based extreme learning machine for network intrusion detection. *IEEE Access*, *9*, 16062–16091.

Wei, W., Ming, Z., Xuewen, Z., Xiaozhou, Y., & Sheng, Y. (2017). Malware traffic classification using convolutional neural network for representation learning. *2017 International Conference on Information Networking*, 712–717. https://doi.org/10.1109/ICOIN.2017.7899588

Wong, W.-K., Moore, A., Cooper, G., & Wagner, M. (2003). Bayesian network anomaly pattern detection for disease outbreaks. *Proceedings of the 20th International Conference on International Conference on Machine Learning*, 808–815.

Xiao, Y., Xing, C., Zhang, T., & Zhao, Z. (2019). An intrusion detection model based on feature reduction and convolutional neural networks. *IEEE Access*, *7*, 42210–42219. https://doi.org/10.1109/ACCESS.2019.2904620

Xiaofeng, Z., & Xiaohong, H. (2017). Research on intrusion detection based on improved combination of k-means and multi-level SVM. *2017 IEEE 17th International Conference on Communication Technology*, 2042–2045. https://doi.org/10.1109/ICCT.2017.8359987

Xu, J., & Shelton, C. R. (2010). Intrusion detection using continuous time Bayesian networks. *Journal of Artificial Intelligence Research*, *39*, 745–774.

Yang, Z., Liu, X., Li, T., Wu, D., Wang, J., Zhao, Y., & Han, H. (2022). A systematic literature review of methods and datasets for anomaly-based network intrusion detection. *Computers & Security*, *116*, 102675.

Yeung, D.-Y., & Chow, C. (2002). Parzen-window network intrusion detectors. *2002 International Conference on Pattern Recognition*, *4*, 385–388. https://doi.org/10.1109/ICPR.2002.1047476

Zoppi, T., Ceccarelli, A., Capecchi, T., & Bondavalli, A. (2021). Unsupervised anomaly detectors to detect intrusions in the current threat landscape. *ACM/IMS Transactions on Data Science*, *2*(2), 1–26.

Appendix A

Example of raw event

```
Event   Time           Source                Destination           Length Dst
Port Information
     15 20.759999      10.0.2.4              10.0.2.100            132    42309
  Tree Connect AndX Response

Frame 15: 132 bytes on wire (1056 bits), 132 bytes captured (1056 bits)
    Encapsulation type: Ethernet (1)
    Arrival Time: Feb 24, 2022 13:28:11.765054000 Hora estándar romance
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1645705691.765054000 seconds
    [Time delta from previous captured frame: 0.000146000 seconds]
    [Time delta from previous displayed frame: 0.000146000 seconds]
    [Time since reference or first frame: 20.759999000 seconds]
    Frame Number: 15
    Frame Length: 132 bytes (1056 bits)
    Capture Length: 132 bytes (1056 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:tcp:nbss:smb]
    [Coloring Rule Name: SMB]
    [Coloring Rule String: smb || nbss || nbns || netbios]
Ethernet II, Src: VMware_15:fd:15 (00:0c:29:15:fd:15), Dst: VMware_bf:32:2f
(00:0c:29:bf:32:2f)
    Destination: VMware_bf:32:2f (00:0c:29:bf:32:2f)
    Source: VMware_15:fd:15 (00:0c:29:15:fd:15)
    Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.100
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 118
    Identification: 0x2909 (10505)
    Flags: 0x40, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0xb911 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.0.2.4
    Destination Address: 10.0.2.100
Transmission Control Protocol, Src Port: 445, Dst Port: 42309, Seq: 814, Ack:
1019, Len: 66
    Source Port: 445
    Destination Port: 42309
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (63)]
    [TCP Segment Len: 66]
    Sequence Number: 814    (relative sequence number)
    Sequence Number (raw): 1408609355
    [Next Sequence Number: 880    (relative sequence number)]
    Acknowledgment Number: 1019    (relative ack number)
    Acknowledgment number (raw): 3775614437
    1000 .... = Header Length: 32 bytes (8)
    Flags: 0x018 (PSH, ACK)
    Window: 256
    [Calculated window size: 65536]
    [Window size scaling factor: 256]
    Checksum: 0xcbdc [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
    Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    [Timestamps]
    [SEQ/ACK analysis]
    TCP payload (66 bytes)
NetBIOS Session Service
SMB (Server Message Block Protocol)
```