UNIVERSITAT POLITÈCNICA DE VALÈNCIA

TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

PROBABILISTIC PREDICTION OF THE BEHAVIOUR OF CYBER ATTACKERS

Pablo Camacho González

Tutor: Pilar Candelas Valiente

Trabajo Fin de Máster presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Máster Universitario en Ingeniería de Telecomunicación

Curso 2018-19

Valencia, 16 de Mayo de 2020

DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2019

Probabilistic prediction of the behaviour of cyber attackers

PABLO CAMACHO GONZÁLEZ

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Probabilistic prediction of the behaviour of cyber attackers

PABLO CAMACHO GONZÁLEZ

Date: May 16, 2020
Supervisor: Pontus Johnson
Examiner: Mathias Ekstedt
School of Electrical Engineering and Computer Science
Swedish title: Probabilistisk förutsägelse av beteenden av cyberattacker

Acknowledgements

Throughout the process of writing this project, I have received a great support that has made this effort easier to carry out. First of all, I'd specially like to thank Mr. Pontus Johnson, who has the title of supervisor of this project, and whose help and patience represented two important keys for a correct development. I'd also like to thank the organisations of KTH Royal Institute of Technology (Stockholm, Sweden) and Universitat Politècnica de València (Valencia, Spain) for the opportunity to develop my academic work, learn and grow as a professional.

I'd like to acknowledge the Systems Architecture and Security (SSAS) research group for the opportunity to develop this project, specially to Nikolaos Kakouros for his support, and Mathias Ekstedt for his interest and willing to help.

In addition, I'd like to thank my parents, my siblings and Miguel, as well as my close friends for their important presence and support in this and all stages of my life. I'd also like to acknowledge all the people who crossed my path during this time in Spain and Sweden, and took an important role in my life, as well as Mariano Villalta and Jon Arrizabalaga for their help in the final stage of this process.

All the people mentioned here, along with many others who became special friends during the journey, represented the best personal and professional support for this project.

Abstract

Probabilistic graphical models are one of the most important tools used to learn from data and make decisions. Data analysis and representation methods are highly used in cyber security as a powerful way to find understanding and improve as cyber crime advances and cyber attacks become smarter. The goal of this project is to analyse and represent the behaviour of cyber attackers by applying a Bayesian network model to a cyber range. In order to learn and understand how this model can be used to represent behavioural patterns, a deep research about probability and graph theory is made, as well as Bayesian theory and analysis. A simulated dataset of detected cyber attacks is filtered and analysed in order to find patterns and trace the actions of cyber attackers. The processed information about dependencies and their probabilities is later used to calculate predictive probabilities, and the designed model is tested with the simulated dataset. The result is an interactive graphical model based on a Bayesian network, which accurately represents the behaviour of cyber attackers.

Sammanfattning

Probabilistiska Grafiska modeller är en av de viktigaste verktygen som används för att förstå sig på data och fatta beslut. Dataanalys och framställning metoder används i hög grad inom cybersäkerhet som ett kraftig sätt att hitta, förstå och förbättra när cyberbrott avancerar och cyberattackerna bli smartare. Målet med detta projektet är att analysera samt presentera beteenden av cyberattackerna genom att applicera en Bayesian nätverksmodell till en cyber räckvidd. För att kunna lära sig och förstå hur denna modellen används för att framställa beteende mönster, utförs en grundlig foskning om probabilistisk och grafteori sävälsom Bayesian teori och analys. En simulerad data set på upptäckta cyberattack filtreras och analyseras för att i sin tur kunna hitta mönster och spåra handlingar av cyberattackerna. Det bearbetade uppgifter om dependencies och dess sannolikheter kommer till användning för att beräkna förutsägbara sannolikheter, den designade modellen prövas med den simulerade datasetet. Resultatet är en interaktiv grafisk modell baserad på en Bayesian nätverk som noggrant vis framställer beteenden av cyberattackerna.

Resumen

Los modelos grafo probabilísticos son una de las herramientas usadas para aprender sobre la información y tomar decisiones. Métodos de análisis y representación de datos son altamente utilizados en seguridad informática, como vía para entender y mejorar a la vez que el cibercrimen evoluciona y los ciberataques se vuelven más inteligentes y elaborados. La meta de este proyecto es analizar y representar el comportamiento de ciberatacantes mediante la aplicación de un modelo de red Bayesiana sobre un sistema de aprendizaje y entrenamiento destinado a procesos de ciberseguridad. Con el fin de aprender y entender cómo este modelo puede ser utilizado para representar patrones de comportamiento, se lleva a cabo una profunda investigación sobre teoría probabilística y de grafos, así como sobre teoría y análisis Bayesiano. Un conjunto simulado de datos, formado por ciberataques detectados, es filtrado y analizado con el fin de encontrar patrones y trazar las acciones de ciberatacantes. La información procesada sobre dependencias y sus probabilidades es posteriormente utilizada para calcular probabilidades predictivas, y el modelo diseñado es testeado con el conjunto de datos previamente simulado. El resultado es un modelo gráfico interactivo basado en una red Bayesiana, el cual representa el comportamiento de ciberatacantes con precisión.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

At this time of human history, communications get wider and easier every day. Connections and systems have been highly developed in the last few decades due to the evolution of internet, and it's the most important field of investigation and research in the early XXI century. Telecommunications advances are being applied to many types of projects and technological fields, making them easier and smarter, and opening a new world full of opportunities. Due to the fast evolution of communication technologies and computer science, many inhabitants of the earth can't handle the fast changes in the society, finding it really difficult to adapt to the new communications and social systems and having a lot of problems with the way they manage to create, hide and transfer their information. Computer science advances are being applied to everything in the daily life of the modern world, from education and healthcare to IoT technologies [1]. This technological evolution has brought many facilities to modern society, but has also created problems and dangers that are the main focus of work and research of many computer science experts [2]. The field of study that focus on computational systems safety and its investigation is called **cyber security**. The definition of this term has always been a difficult task due to its big scope, in which this project proposal is included, and big efforts have been made in order to find a proper one, like it's described in [3]. In general terms, all its definitions refer to the **protection and safety of systems and networks**.

Despite all the efforts made to construct a totally safe system, not a single one can be defined like that yet. Every system has a vulnerability and all the protocols and cyber security techniques start becoming more unsafe as time passes since the first time they're applied. Cyber security is constantly evolving, finding vulnerabilities and threats, and building methods to protect and

avoid risks. Identifying threats is an important part of this process, which allows to find cyber attackers behavioural changes by analysing old reports [4]. Some organisations like **ENISA** keep working on this field to keep track of the **threat landscape** and gather information. Latest cyber attack trends can be found in [5][6]. Many countermeasures have been developed against the most common cyberattacks like Brute Force, Denial of Service (DDoS) or Phishing [7], as well as more complex defense models and attack analysis methods like **Defense Trees** [8] or **Attack Countermeasure Trees** [9]. Information about cyber attacks is an important factor to extract behavioural data from detected events, for which methods like **reverse engineering** are widely used in order to learn from them. Probabilistic methods are a powerful tool to learn from data, and although some solutions regarding to cyber security have been proposed, there's still so much to learn about probabilistic cyber attackers behaviour prediction and understanding.

Probabilistic graphical models are widely used in fields where reasoning and learning efficiently from raw data are needed, like Artificial Intelligence [10], as well as to build **decision support** systems, for which many different solutions applied to different fields can be found [11][12][13]. This project proposes a solution to learn from cyber attacks data by applying a probabilistic graphical model based on **Bayesian Networks**, in order to trace patterns, deal with uncertainty and predict cyber attackers behaviour inside a network system. In order to build a reliable model, a solution must be found for three main issues:

1. **Filtering** the dataset of detected cyber attacks in order to find the events (Nodes) that compose the graphical model, as well as finding dependencies between them by applying **Structure Learning** methods.

2. Reasoning and calculating **Conditional probabilities**, and use them to compute the **inference** of any event given all the possibilities, in order to predict and find out how likely are all the possible future events to occur given a certain situation.

3. Building the graphical model and tracing the calculated patterns, allowing to analyse and understand them.

The main goal of all this process, described in this draft, is to create an efficient and scalable tool to learn from data, providing an specific approach of a reliable probabilistic model solution for **cyber security**.

Chapter 2

Background

To understand the scope of this work, which is closely associated with data analysis for cyber security, it is necessary to expose some background about this field. Understanding how a system can be compromised is the most important factor for cyber security to prevent attacks and develop safety programs. Summarizing, the study of cyber attackers behaviour is crucial to minimize threats and protect data, and it has a whole world of science inside, for which the presence of probabilistic models represents an important source of information. After detecting an attack, analysis methods are used to understand how it was executed, being able to elaborate a forensic report and find ways to avoid being affected by a similar action. Many other research reports and analysis methods have been developed by researchers and students, using modern science methods to develop algorithms and improve the way we understand the information. In order to learn how probabilistic models are usually applied to cyber attacks tracing and prediction, and be able to elaborate a reliable model, it's important to study and understand what has been found by researchers, and the solutions that have been proposed.

Prediction is one of the keys on which many cyber security methods are focused. In fact, almost all the efforts made in order to avoid risks in a network system need a reliable threat prediction method [14]. The research made in [15] shows that cyber threat prediction modeling techniques can be divided in two groups: **Statistical modeling** and **Algorithmic modeling**. Our prediction model proposal can be included in the second group, due to its relation with **probabilistic modeling**. This type of techniques include probabilistic algorithms and usually have results that are easy to manage and understand. Although there's a common drawback when applying one of these methods, which is that it's difficult and confusing to find proper **prior** values (Usually

needed in methods like **Bayesian networks**, described in section 3.3.6), they represent one of the most efficient options for discrete information. Probabilistic methods like **Markov chains** or **Bayesian Networks** are the most usual algorithms used in prediction techniques, being applied for many different solutions with different inputs, like in the case of [16] where a probabilistic algorithm is applied using the information of single entities in order to compute the **likelihood** of malicious behaviour coming from them. Many other solutions for detection and prediction are being developed nowadays, like it can be observed in research reports like [17] where a deep literature analysis about research trends in detection and prediction for **insider threats** is made, or [18] where the most used methods for prediction and forecasting in cyber security are analysed, comparing continuous and discrete methods and showing the strong presence of **Probabilistic graphical models** for techniques where discrete data is involved, like it's the case presented in this project. Other graphical solutions that are not associated with probability theory can also act as a powerful tool for data reasoning, like the proposal made in [19] to build an ontology for **cyber security knowledge**, like the example showed in figure 2.1.

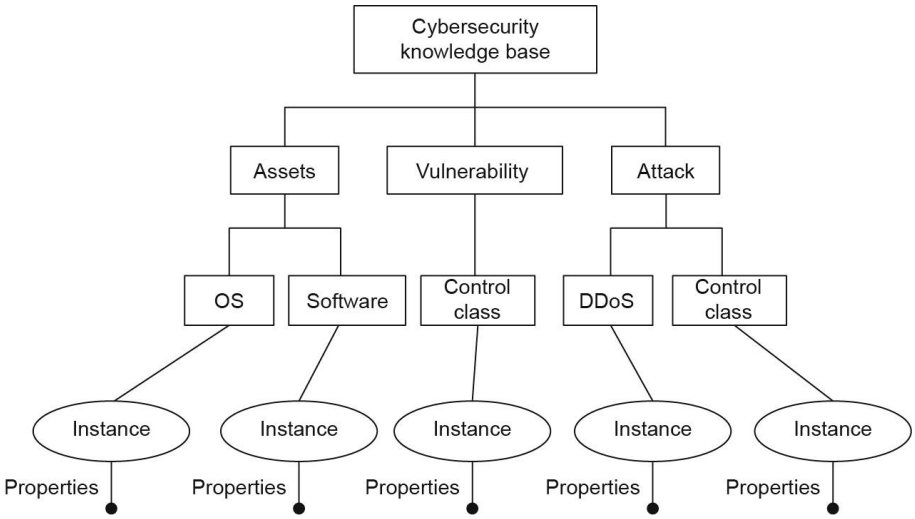

Figure 2.1: Knowledge base scheme example [19]

The main goal of the model developed and described in this project is to reduce the uncertainty by tracing and analysing cyber attackers behaviour patterns inside a network system, applying a probabilistic graphical model based in **Bayesian networks**. Bayesian networks represent an efficient tool for reasoning and representation, and have an strong presence in **Prediction for cyber security**. There's a high number of solutions and proposals involving this

model, applying **Bayesian theory** to different data with different purposes associated with cyber security, like the project developed in [20] to build a risk model based on Bayesian Networks, applied to a real system. There are many other interesting solutions closely associated with Bayesian networks and the model proposed in this draft, but applying a different approach for probabilistic prediction. Some proposals like the model for attack plan recognition described by the authors of [21], which goal is to find attack scenarios and predict cyber attackers strategies using security alerts and causal networks, or the model proposed in [22] for cyber attacks prediction using unconventional signals, can be included in this same scope, although these models propose possible solutions using data that can be generated **before** an attack is performed, instead of learning from performed actions. A very interesting approach proposed by Jinyu Wu, Lihua Yin and Yunchuan Guo in [23] also includes Bayesian networks for reasoning to find vulnerabilities in specific environments, as well as specific information about the network, like **Information value**, **Usage condition** and **Attack history** in order to predict cyber attacks. Although this proposal is interesting, Bayesian analysis is applied for information reasoning in this case, and not for cyber attackers patterns analysis.

Appart from research reports and proposals, an important part of the knowledge associated with this type of models is also present in specific tools for probabilistic prediction like **P2CySeMoL**[24], which is a **Predictive, Probabilistic Cyber Security Modeling Language** that provides an attack graph tool for enterprise architectures. This project was developed at **KTH Royal Institute of Technology** and represents an interesting approach and a useful and smart way to apply probabilistic analysis methods.

The very nature of a project like the one approached in this case requires an special environment with specific properties. In this case, the research is mainly focused on data reasoning, prediction and representation, but understanding and learning about the properties of the scenarios that are being analysed also represents an important and interesting point of view. The success of an algorithm that is focused on learning from data relies upon the good performance of an strong detection system. Attack detection and understanding represents a very important task, for which many efforts are made. An interesting approach to understand the importance of these efforts is the one focused on modern threats, for which new algorithms and researches are being constantly developed, like the example of [25] where a descriptive approach about **Phishing URL detection** through top-level domain analysis is presented, or the case of **MaldomDetector** [26], which represents a detection system for algorithmically generated domain names. Another interesting ap-

proach regarding to cyber attack detection applied to modern real systems is [27], where the authors present a control scheme proposal for **False Data Injection** attacks in networked control systems, or the model proposed in [28] for cyber attack detection in IoT systems using distributed deep learning. As it was mentioned before, no system is completely safe, which also applies to **Intrusion Detection Systems**. An interesting example of this approach is [29], where the authors propose an attack detection algorithm based on machine learning, focused on countering attacks performed against IDS.

The model described in this draft is based on a probabilistic prediction algorithm, which goal is to learn from data in order to find behavioural patterns. When talking about cyber security algorithms, probability and strategy are closely associated concepts. It is usual to find cyber security models based on **Game theory** for defense [30] and decision-making algorithms, like the proposal made in [31] of an stochastic model of cyber attacks with imperfect detection. Defense and attack models for networks represent an extended research field, as it can be observed in [32], where a review of attack and defense models for systems is made. An specially interesting point of view for this proposal is the application of Bayesian networks for defense models, like the one proposed in [33] for optimal strategies, which shows the different possibilities of probabilistic graphical models applied to different cyber secrurity approaches. From an strategic point of view, a probabilistic graphical model for data reasoning like the one proposed in this draft represents a promising method to **learn** and **understand**, and can be an interesting addition for the cyber security research field.

Summarizing, probabilistic graphical models for cyber security analysis and prediction methods are widely used in different ways and ranges, becoming an interesting and important part of the research in this field and improving the way information is processed, proving versatility and efficiency. This project offers a solution proposal including **Bayesian networks** for cyber security analysis, not focused on the safety of a certain system, but targeting the analysis and prediction of the behaviour of cyber attackers, learning from their actions to build models of the patterns they follow. This problem statement leads to the following research questions:

1. Can the behaviour of cyber attackers be accurately modelled by applying Probabilistic graphical methods?

2. Do Bayesian Networks represent an efficient and reliable model for tracing and prediction using behavioural data?

The answer to the questions stated above represents the main goal of all the process described in this draft. The result of these efforts is expected to bring the knowledge needed to understand the problems and solutions presented here.

Chapter 3

Theory

The method for probabilistic prediction proposed in this draft is based on the use of Bayesian Networks to trace cyber attackers behaviour, using detected attacks as events and defining the relations between them. Bayesian networks are based on **Bayesian probability theory**, being defined as a model for causality computation by applying its theoretical concepts. To understand the process of calculus and construction of a Bayesian network, it's necessary to define important concepts about **relations between events** and **probability**, as well as the properties of the resulting graphical model.

3.1 Graph theory

Essentially, Bayesian networks consist of a type of graphical model called **Directed Acyclic Graph**, often called **DAG**. Basically, a DAG is a graph with special properties, and it's important to understand the basic concepts of graph theory in order to know what it represents.

Like in this case, it is usually necessary to represent and model flows and relations when using mathematics to solve a problem or add some understanding about a situation. A relation between two or more objects is represented using **graphs**. A graph consists of a structure made up of **vertices** and **edges** (Lines connecting nodes) [34], which can be mathematically expressed as:

$$G(V, E) \tag{3.1}$$

A vertex can represent a situation, state or an element with defined properties, connected with other nodes by edges which represent an existing relation between them. A graph can also be defined as a set of **subgraphs** which vertices and relations are included in its set range. This concept perfectly matches

the nature of a cyber attacker behavioural pattern, which can be considered as an attack graph where smaller sets in the same set can also represent attack processes [35].

An attack pattern can be represented as a series of events with causal dependencies, creating a graph model associated with a **tree** topology, like the example in figure 3.1. This type of topology fulfils the **acyclic** requirement of a DAG.

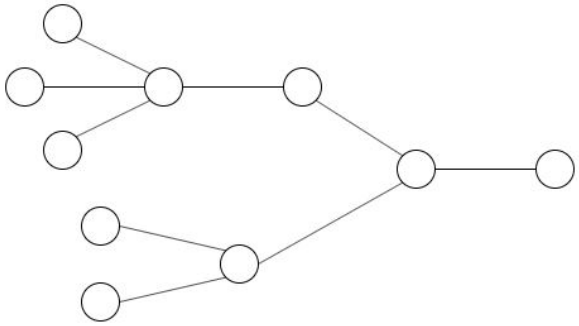

Figure 3.1: Tree topology example

In a cyber attack process, the concept of **reciprocity** loses its sense when analysing actions performed by one host to another. Usually, the actions performed by a cyber attacker represent steps of a process, with an objective and direction, and this can't be modelled using **undirected** models. When representing a cyber attacker behaviour pattern, the direction taken by every action has a high importance for the context understanding of the step, and makes it necessary to use **directed** models. A directed graph showing dependencies between vertices, like the one in figure 3.2, allows to understand a process like this.

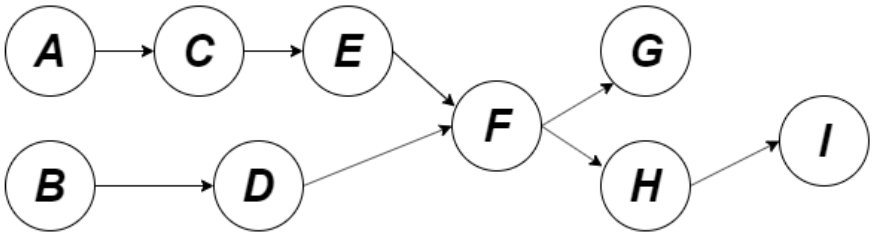

Figure 3.2: Directed graph example

As it was defined at the beginning of this section, a directed graph with no cycles is the type of graph used in Bayesian networks (Directed acyclic

graph), and represents a suitable model for this case, now that the concepts of **directed** and **acyclic** are considered understood. Every directed connection in a DAG represents a causal relation where the source node is called **parent** and the target is its **child** [36], and tere's no possible way that a process started in a certain node can end up in the same one by following the right direction of the edges.

3.1.1 Directed Acyclic Graphs for cyber security

Directed Acyclic Graphs are often used in applications where **precedence dependencies** are present, like project temporal schemes, Software design processes or hierarchical structures in general [37]. To calculate dependencies in this case, the precedence factor is also relevant to define the steps of an attack process. The expected result of this project can be associated with **Directed Tree topologies**.

Other solutions using this type of DAGs for cyber security have been developed before, and represent an interesting approach for this project, like the cyber attack analysis model described in [38] which was published in **Dr. Dobb'r Journal** in 1999, where the author Bruce Schneier presents a method capable to define the security of a system.

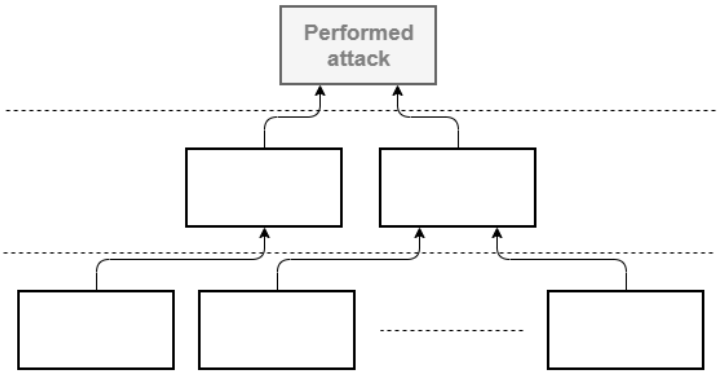

Figure 3.3: Attack tree graph [38]

Like it was described in section 2, Tree topologies have been widely used for cyber security analysis, and share many properties with this proposal due to the use of DAGs. Although the type of graphical model is the same, the theoretical approach is different, and the result of our algorithm can't be totally included in the scope of this topology, being this just a possible outcome of the resulting network.

3.2 Probability theory

3.2.1 Introduction

The behaviour of different cyber attackers inside the same system usually follow some kind of **pattern** and have many characteristics in common, and that's why studying the behaviour of cyber attackers and create attack models can increase the efficiency of a security system. Analysing the steps made by attackers is useful to learn how to avoid the same attack in the future, but systems usually have more than one vulnerability, making this not enough to protect them. It's not unusual to find different attack processes leading to the same target, as well as different events who share a same previous action. Computing the probability weight based on observed data for future events of, for instance, a discrete distribution of possible outcomes, is a typical application of **probabilistic forecasting** [39], and it's present in the steps analysis of this model.

The construction of Bayesian Networks is based on Bayesian probability theory, and it's necessary to define the basic concepts of probability in order to fully understand the meaning of the results of the model.

3.2.2 Probability

The term **probability** refers to a numerical figure between 0 and 1 that defines how likely is an event to occur, and the term **probability theory** defines the science and methods that treat probability in a rigorous mathematical way [40]. This mathematic branch is based on the analysis of random phenomena. Random events, whose result of a single trial can't be predicted with certainty but present some kind of regularity when the number of samples grows, can be analysed using the knowledge about same events that occurred in the past, finding patterns and predicting the probabilities for all possible outcomes. This theory is based on three axioms that define all the science inside [41].

1. For any event A, the probability of A is greater or equal to 0.

$$P(A) \geq 0 \tag{3.2}$$

2. The probability of, at least, one of the possible outcomes of an event to occur is 1.

$$P(S) = 1 \tag{3.3}$$

3. If A and B are mutually exclusive outcomes, the probability of either A or B happening is the probability of A happening plus the probability of B happening.
$$P(A \cup B) = P(A) + P(B) \tag{3.4}$$

Study and predict the actions of cyber attackers supposes a difficult task due to the variety of actions and the ignorance about possible threats that haven't been discovered yet. Every possible action must be taken seriously, and the probabilistic analysis of the gathered data from cyber attacks brings a big amount of knowledge, which can be used to find models and patterns. The actions made by a cyber attacker can be seen as a **sample space** of a discrete distribution (A finite and countable set of results) where every one of them has a probability of success, and the sum of all of the probabilities is equal to one (Which means that every probability has a value between zero and one).

$$f(x) \in [0, 1] \text{ for all } x \in \Omega \tag{3.5}$$

$$\sum_{x \in \Omega} f(x) = 1 \tag{3.6}$$

When a sample scope has a big number of results, probabilities start to converge inside a different interval of values for every event. This makes it possible to find patterns and to create attack models analysing the steps taken by cyber attackers, being responsive to changes and more accurate as data increases.

3.2.3 Random variables

The concept of random variable refers to a function that is directly associated to a **random experiment**. The result of a real-valued random variable function given an experiment outcome is contained in a **sample space**, where the variable is defined. Generally, the probability of an event is stated as the probability of the involved random variable to be present in the interval (a,b) given the outcome of an event [41][42].

$$P\{\omega : a \leq R(\omega) \leq b\} \tag{3.7}$$

Every event involving a random variable is associated with a value of that random variable in a set **B** called **Borel subset**, which is considered an event that belongs to the sigma field **F**. The outcome for every possible input of an experiment is defined by a continuous random variable (Infinite range),

and continuous random variables properties are defined by two main functions [43]:

- **Cumulative distribution function (CDF):** This function represents the continuous probability distribution for a set of samples of a random variable.

$$F_X(x) = P(X \leq x) \tag{3.8}$$

  A random variable is considered continuous when its CDF is continuous for all values of x.

$$P(a < X \leq b) = F_X(b) - F_X(a) \tag{3.9}$$

- **Probability Density function (PDF):** This function defines the probability density of a continuous random variable for a given sample.

$$f(x) = \lim_{\Delta \to 0^+} \frac{P(x < X \leq x + \Delta)}{\Delta} \Rightarrow f(x) = \lim_{\Delta \to 0^+} \frac{F_X(x + \Delta) - F_X(x)}{\Delta} \tag{3.10}$$

  For a random variable with an absolutely continuous CDF, the PDF is defined as a derivate from the distribution function.

$$f_X(x) = \frac{dF_X(x)}{dx} \tag{3.11}$$

A random variable with a finite and countable range of samples is called **discrete random variable**. In this project, cyber attacks are considered discrete distributions of random variables, which means there's a finite and countable range of samples that refer to successful results (or not) of the actions.

3.2.4 Algebra of events

The classical definition of probability defines the probability of an event as the "number of favourable outcomes divided by the total number of outcomes, where all outcomes are equally likely" [41]. Probabilities are assigned to events, and samples spaces of events have different meanings depending on the nature of the variables. New outcomes are created from old ones using different operations:

- **Union:** The concept of Union refers to the set of points belonging to, at least, one of the events. The third axiom of probability theory.

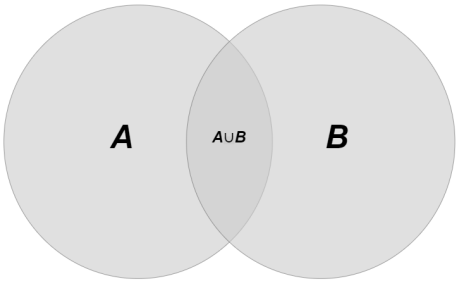

Figure 3.4: Union

- **Intersection:** Set of samples belonging to all the events.

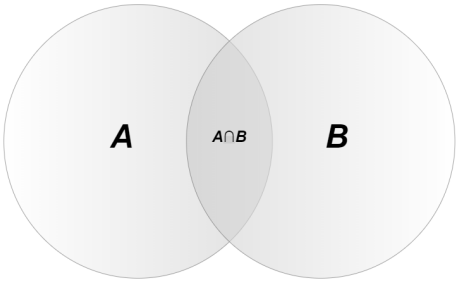

Figure 3.5: Intersection

- **Complement:** The complement of an event refers to all the points that don't belong to that event.

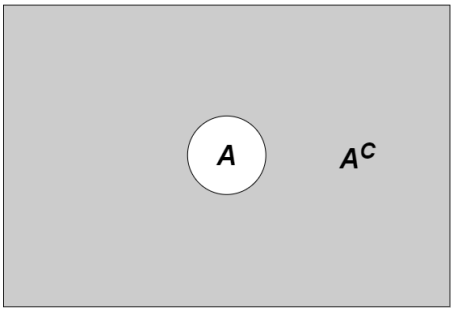

Figure 3.6: Complement

Understanding these operations is a key factor when analysing samples spaces of attack outcomes. Finding dependencies between actions requires the usage of different probability concepts, like **conditional probability**, which are based on this theory. It is usual that a cyber attack is associated with one or

more actions, leaving fingerprints and useful information in different parts of a dataset. The **Theorem of Total Probability** defines the method to compute the probability of an event to occur as a sum of all its possible intersections with other mutually exclusive events, and adding marginal probabilities, following the property of **joint probability** for mutually exclusive events. This concept is closely asociated with the mathematical expression of a Bayesian Network.

$$P(A) = \sum_i P\left(A \cap B_i\right) \Rightarrow P(A) = \sum_i P\left(B_i\right) P\left(A|B_i\right) \tag{3.12}$$

3.2.5 Expectation

The **expected value** of a random variable, given a long list of samples, is defined as the convergence value of a large number of samples that come from the same experiment. In case of a **finite** discrete random variable, the expected value is equal to the sum of x times the probability of a random variable being equal to x [44].

$$E(R) = \sum_x xP\{R = x\} \tag{3.13}$$

In this case, the definition of expected value loses its validity when trying to apply it to an absolutely continuous case. The convergence of independent values can only be studied and computed given a discrete distribution, and it becomes useful when a dataset is big enough and marginal probabilities of, in this case, cyber attacks, can be used to find dependencies and build an attack model.

3.2.6 Joint probability

The intersection of two or more events (Represented in section 3.2.4) is also known as joint probability, or the probability of events A and B to occur. This definition refers to discrete variables, and all the possible outcomes for the joint probability of all the observed values of two discrete variables is called **Joint probability distribution**. The probability of two events happening is highly important to study and calculate dependencies between cyber attacks because of its relation with **conditional probability**, and it's present in the whole process of analysis and representation.

3.2.7 Conditional probability

It is said that two events are dependent when the occurrence of one of them affects the occurrence of the other. Conditional probability is the term that defines how strong the dependency between two events is, so the probability of an event A given an event B would be computed taking in account all the times when A occurs and B have occurred, which can be deducted dividing the intersection between events A and B by the marginal probability of event B.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{3.14}$$

Applying the definition of joint probability for non-exclusive variables, it's possible to obtain the **Bayes theorem** [45], which is used for diagnosis and forecasting because of the possibility to predict the posterior probability of any event given a **prior** and an **evidence** factor.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{3.15}$$

Conditional probability and Bayes' theorem are a very important part in the process of tracing and predicting cyber attacks in this case, allowing to create a model which can accurately predict the possible outcomes of an action given the past experience.

3.3 Bayesian probability theory

3.3.1 Introduction

The creation and functioning of a Bayesian network is based on a **Bayesian** approach for probability theory. Describing the basic concepts of this field is necessary to understand the nature of this probabilistic model, and be able to describe the result of this project. As it was defined before, probability theory treats probability in a mathematical way based on three axioms that describe the conditions and nature of probability. In this case, Bayesian probability is based in **Bayes' rule**, which describes the nature of **conditional probabilities**. The concept of probability is not interpreted as something associated with frequency or propensity, but more like the level of certainty or truth related to a possible outcome. In conditional probability, the joint probability of two events is defined like:

$$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A) \tag{3.16}$$

Bayesian analysis allows to calculate probability using a **hypothesis** and combining it with real data. A hypothesis is a consideration of how the analyst thinks the outcome of an event will be. It's important to consider that this is created from evidence only, expectation or both, and that's why it results to be the most subjective part of a Bayesian analysis experiment. This element has a probability ***P(H)*** called **prior**. The other two elements of Bayesian theory are ***P(E)***, which is the result of integrating the joint probability of evidence and hypothesis ***P(E|H)P(H)*** over ***H***, called **normalization constant**, and the term called **posterior**, which is the result of the **likelihood** function, the conditional probability of ***H*** given ***E***, represented by ***P(H|E)***.

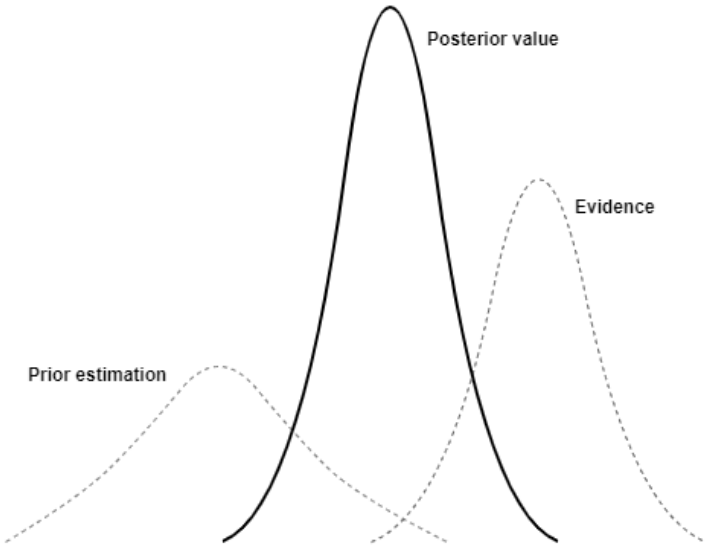

Figure 3.7: Graphical representation of the Bayes theorem factors

Bayesian probability theory represents the evolution and learning of an expected outcome mixed with real data. The fact that a prediction result has a better performance with more information brings an interesting point of view, as well as a powerful tool to start developing and create reliable enough attack models, making them responsive to changes, easy to adapt and better to understand. As it was explained before, what makes Bayesian theory so interesting and useful is that it uses the information about what is **not known** (Belief and Uncertainty).

3.3.2 Applications

Bayesian probability is widely used for applications that need some kind of information processing. Generally, a Bayesian theory application has an **input** that gives context and conditions, and an **output** corresponding to the processed information given as an input. The input of a Bayesian theory application usually consists in sets made out of measurements, laws and constraints that describe some kind of system or world, and are processed to calculate **predictions**, understand the systems and make **decisions**. Many of the applications for which Bayesian theory is the core of the model are associated with parameter estimation, prediction models, decision making, forecast, patterns searching and hypothesis. Bayesian theory seems to be very appropriated for the creation of a cyber security diagnosis and prediction model. The problem of dependencies searching and attack prediction suits very well in an input-output format, allowing to perform accurate calculations and predictions with the available datasets from systems monitoring.

3.3.3 Axioms

The Bayesian approach for **probability theory** is focused on uncertainty calculation and hypotheses. The main difference with the classical definition of probability theory is that the Bayesian approach doesn't see probability as **frequency** of events, but as certainty/uncertainty. The rules for certainty calculus, presented by **Cox** [46], define the application structure basic relations for Bayesian probability theory.

1. The joint probability of two elements ***A*** and ***B*** is a function of what is known about ***A*** and what is known about ***B*** given ***A***.

$$P(A, B) = f\{P(A), P(B|A)\} \tag{3.17}$$

2. The negation function of the negation of the probability of the element ***A*** is equal to the probability of ***A***.

$$g(g(P(A))) = P(A) \tag{3.18}$$

3. The negation function of an operation **OR** between two statements is equal to the operation **AND** of the negation function of both statements.

$$g(P(A \,\mathrm{OR}\, B)) = g(P(A)) \,\mathrm{AND}\, g(P(B)) \tag{3.19}$$

This approach is further studied and built by Jaynes, who stated that Bayesian probability and its plausibility is defined by three main "desiderata" [47]:

- **Sum rule**
$$P(x + y|H) = P(x|H) + p(y|H) \tag{3.20}$$
- **Product rule**
$$P(xy|H) = P(x|yH)P(y|H) \tag{3.21}$$
- **Bayes' rule**
$$P(\text{Model}|\text{Data}, H) = \frac{P(\text{Data}|\text{Model}, H)}{P(\text{Data}|H)} P(\text{Model}|H) \tag{3.22}$$

These rules are important tools to compute the probability information of relations between events, key for the calculus of **inference**.

3.3.4 Information

The available information about every part of a Bayesian reasoning system must be made understandable in order to be interpreted by the users. Usually, the information content of a PDF is processed and turned into a simple scalar number, which must describe the very nature and uncertainty of that part of the system.

Information function

The information function, which properties were proposed by **I. J. Good** in 1966 [48], defines the information of a model $\boldsymbol{M}$ given an evidence information $\boldsymbol{E}$ and a context $\boldsymbol{C}$.

$$I(M : E|C) \tag{3.23}$$

One of the properties derived by Good is that this is an **strictly increasing function**, which means the information quantity increases when any of its arguments does.

Entropy

The concept of entropy refers to the measure of the **uncertainty** of the state of a system. The measure of information was presented by **Shannon** as a measure to quantify the transmission capacity of communication channels,

which is a qualified way to compute the average measure of the information. The name of **Entropy** comes from its similarity with the uncertainty measure of thermodynamic systems. The axioms that define the entropy of a probability distribution are [49]:

1. An entropy function $\boldsymbol{H}$ is a continuous function of $\boldsymbol{p}$.
2. $\boldsymbol{H}$ becomes a **monotonically increasing** function if all the probabilities of the distribution are equal.
3. Uncertainty $\boldsymbol{H}$ does not change depending on order or groups of samples.

Entropy and quantity of the information defined by Shannon for a discrete variable with $\boldsymbol{n}$ samples and probabilities, with an associated information $\boldsymbol{I}$, is the sum of the associated information for every sample.

$$I_n = p_n \ln p_n \quad \Rightarrow \quad H[X] = -\sum_{i=1}^{N} p_i \ln p_i \tag{3.24}$$

Uncertainty allows to calculate the **mutual information** of two variables, which is defined as the observed information of a variable $\boldsymbol{X}$ observing the variable $\boldsymbol{Y}$.

$$I[Y, X] = H[X] - H[X|Y] = H[Y] - H[Y|X] \tag{3.25}$$

In this case, the information observed from a network monitoring is formatted into discrete distributions. Understanding what uncertainty and mutual information means is very important when modelling a reasoning system for cyber security.

3.3.5 Markov chains

In section 3.1, basic concepts of graph theory and its usage in cyber security are defined. When working with uncertainty, generating reliable graphical representations can't be based on fixed values. Found dependencies between states can be stronger than others, and it is necessary to talk about **Probabilistic Graphical models** [50]. Probabilistic graphical models represent random variables as states and their dependencies. **Markov chains** represent the probability distribution of transitions between states, describing the behaviour of a system given the information between nodes. This type of graphical model is

based on all the possible **states** of a system, which depend on the number of nodes and all the possible values they can take.

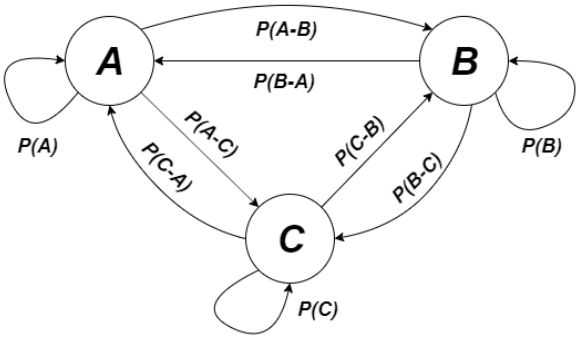

Figure 3.8: Example of a Markov chain

The probability distributions that represent every transition and state are defined in the **transition matrix**, which is the information core of a Markov chain. For a Markov model of ***N*** states, there's an ***NxN*** transition matrix that represents the probability of every possible state of the system.

$$P = \begin{bmatrix} P(A) & P(A-B) & P(A-C) \\ P(B-A) & P(B) & P(B-C) \\ P(C-A) & P(C-B) & P(C) \end{bmatrix} \tag{3.26}$$

This type of model is considered one of the most basic probabilistic graphical models. In a Markov model, every state is considered a binary discrete variable, being able to represent dependencies for two possible states and allowing cycles, representing full relationships between states. A model using **binary discrete variables** can be considered an efficient way to represent cyber attacks as an state with two possible outcomes (Success or failure), but a Markov chain model doesn't allow to represent enough information about joint and conditional probabilities, and neither to study the behaviour of the information when it is affected by many other factors. To represent dependencies and relations and be able to calculate and update **beliefs**, as well as to apply Bayes theorem to calculate inference and predict, it is necessary to apply a Bayesian graphical model, mostly known as **Bayesian network**.

3.3.6 Bayesian networks

Some graphical models, present in basic graph theory, are a good way to represent states and dependencies between them. In order to analyse the behaviour

of cyber attackers inside a system, it is necessary to deal with uncertainty due to the wide interval of possibilities about attacks and dependencies, and be able to **trace** and **predict**. The graphical models used to calculate and represent uncertainty are **Bayesian networks** [51]. This type of model consists of a DAG (Directed Acyclic Graph, section 3.1) made out of **nodes**, that represent random variables, and **edges** that represent dependencies between them. As it was explained before, Bayesian networks are a type of DAG, which means every edge is directed, creating **parental relations** between nodes, and none of the possible ways in the model can be a **cycle** (End up where it started).

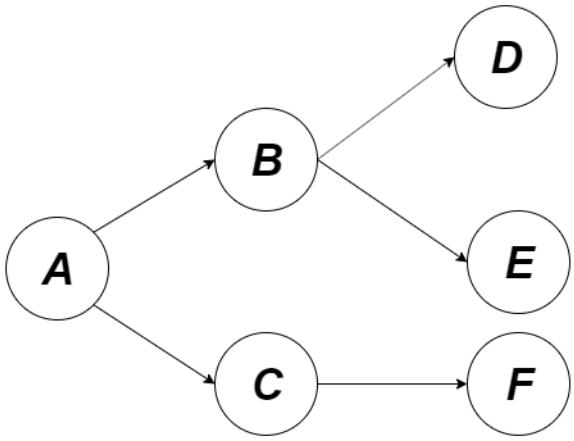

Figure 3.9: Graphical example of a Bayesian network

A Bayesian graphical model represents the probability of a certain number of variables (Nodes), which is affected by their parent nodes. Directed edges represent dependency relations, and the probability of a Bayesian network model is the result of computing all the possible probabilities in the model given their parents.

$$P(X) = \prod_{i=1}^{n} P\left(X_i | \text{ parents } (X_i)\right) \tag{3.27}$$

For instance, the probability of the model in figure 3.9 given the present nodes and edges can be calculated as follows.

$$P(A, B, C, D, E, F) = P(A)P(C|A)P(B|A)P(D|B)P(E|B)P(F|C) \tag{3.28}$$

The probability of every node with a set of parents bigger than 0 is affected by the previous steps, which makes it possible to represent **chains** of connected

events and their relation. This is called the **chain rule of probabilities**.

Explanation

When developing a reasoning system and trying to find certain conclusions from data, like it is always done with Bayesian networks, the main goal is to transmit knowledge finding the best way to make other people understand what is being transmitted. Explaining the reasoning about a system and describing it providing **understanding** define the concept of **explanation** [52]. This definition comes from the approach of **scientific explanation**, but in the case of Bayesian networks and probabilistic models, a more suitable approach is the **explanation in artificial intelligence**, which focuses in the importance of the explanation in **decision-support** systems, which can never be a substitute for the decision of a human being but can be very useful to provide advice, and that's the importance of an accurate and well performed explanation when it comes to this type of systems. The concept of explanation for Bayesian networks is wider than for many other prediction models. In this case, it provides **evidence** explanation and **prediction**. The concept of explanation for Bayesian networks is divided in three categories that summarize the whole task.

- **Explanation of reasoning:** Refers to the explanation of the whole reasoning process of the model and how does it come to the final conclusions.
- **Explanation of model:** Representation of the knowledge of the network in an easily understandable way.
- **Explanation of evidence:** Explanation of the state of the variables, what is observed and their state using the other variables in the network.

Conditional independence

The joint probability distribution of two random variables is directly associated with their conditional probability (Section 3.2.6). In the case of Bayesian networks, the concept of **conditional independence** is a key factor when calculating the probabilities of the nodes. It is said that two variables are conditionally independent when their joint probability is the product of both marginal probabilities.

$$P(A \cap B) = P(A)P(B) \tag{3.29}$$

Given this property, in case of conditional independence, the conditional probability of an event given another event which is not a parent in the model is equal to the marginal probability.

$$P(A|B) = P(A) \tag{3.30}$$

$$P(B|A) = P(B) \tag{3.31}$$

When calculating dependencies in a Bayesian network, the process of **marginalization** consists of finding which variables are not directly associated with others (Are conditionally independent), in order to remove false relations that could intoxicate the calculus. The marginal probability for any event in a Bayesian graphical model can be computed using rules of standard probability [53].

$$P\left(X_i\right) = \sum_{X_1} \cdots \sum_{X_{i-1}} \sum_{X_{i+1}} \cdots \sum_{X_N} P\left(X_1, \ldots, X_N\right) \tag{3.32}$$

D-separation

When talking about conditional independence of events in a Bayesian network, the theory that defines how the evidence about an event affects the other is called **d-separation**, where **d** stands for **dependency**. Two events are d-separated when their conditional probability is not causally affected by the other, and the fact that one of them is happening doesn't give information about the occurrence of the other.

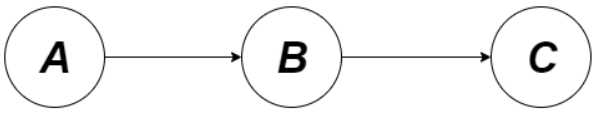

Figure 3.10: Nodes in serial connection

In case of a serial connection, d-separation is impossible for any event. Although two events are not directly connected and don't have a parental relation, the conditional probability of the last event in the chain is causally affected by the events happening before its direct parents.

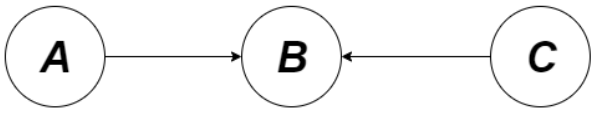

Figure 3.11: Convergent connection

When an event is affected by two different events, they are not necessarily affected between them. In the case of the serial connection in figure 3.11, events ***A*** and ***C*** are d-separated. The same happens when the structure of the connection doesn't show a direct relational way between two events, like it's the case of 3.12.

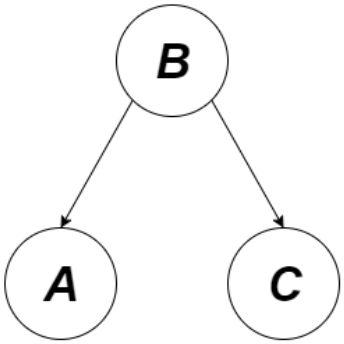

Figure 3.12: Tree connection

Markov blanket

The set of nodes connected with a certain event is called **Markov blanket**. The Markov blanket of an event contains all parents and sons of that event, and all those events that share a child with it.

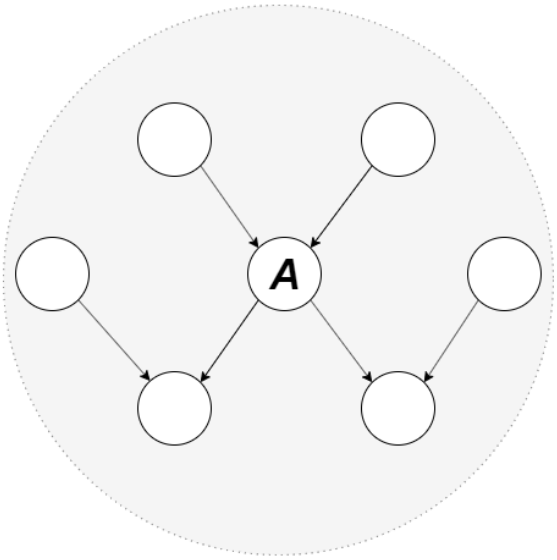

Figure 3.13: Markov blanket

An event is considered **d-separated** of all the events out of its Markov blanket. This concept is especially important to determine conditional probabilities and apply inference methods.

Understanding conditional independency and its properties is of critical importance to calculate and build accurate and reliable patterns in this case.

Structure learning

In order to construct a model that defines the behaviour and characteristics of a system, like cyber attacker patterns, it is necessary to represent states as random variables (Events) and their dependencies. Finding the best graphical model for a Bayesian network is called **structure learning**. This theory has two main different approaches, **score-based** and **constraint-based**.

1. **Score-based approach:** This approach is divided into two steps. The first step is to evaluate how a Bayesian model fits the data and establish a criterion about it, and the second step is to use that criterion to search in the space of all the possible **DAGs** in order to find the model with the maximum score, which is considered the best structure for the case. The score of a Bayesian network given a model $\boldsymbol{BN}$ and data $\boldsymbol{X}$ is computed using the **log-likelihood** of the data in the given model, being the parameters of the model calculated under the principle of **Maximum Likelihood estimation**. The score function avoids overfitting by adding

a second term, computed using the number of samples and the number of parameters in the network [54].

$$S\,\mathrm{core}(BN : X) = LL(BN : D) - \phi(|X|)\|BN\| \tag{3.33}$$

Another function that is widely used to calculate the score of a Bayesian network is the **Bayesian Dirichlet score** [55], which uses the conditional probability of the data given the model and a prior probability for the parameters. This operation is quite similar to the Bayes theorem method for inference calculation.

The most famous algorithm based in this approach is the **Chow-Liu Algorithm** [56]. This algorithm uses the maximum likelihood as the score of the model, using it as a reference to find the best structure, where every node has one parent at most. The process is divided in three steps.

(a) Computing the mutual information for every pair of nodes, creating an undirected graph with weighted edges.

(b) Finding the tree with maximum weights, removing the less-weighted edges of the graph.

(c) Treating every node as a **root**, assigning directions to the edges outward from it, transforming the graph into a directed one. In this case, the direction of the edges is not taken in account due to the symmetry of mutual information and the absence of more than one parent.

2. **Constraint-based approach:** This approach finds the best graphical model given a set of constraints that define dependency relations [57]. The constraints are set making an **independence test**, and the structure of the model is found looking for a DAG that satisfies all of them in the best possible way. Other structure learning approaches and algorithms are being developed by researchers, using alternative methods like **A* search** [58].

Conditional probability table

In a **conditional probability table**, all the knowledge about the probability evidence of a node is gathered and represented. Every node in a Bayesian network has an associated **CPT** with the probabilities of every possible state of the event, given all the values that it and its parents can take.

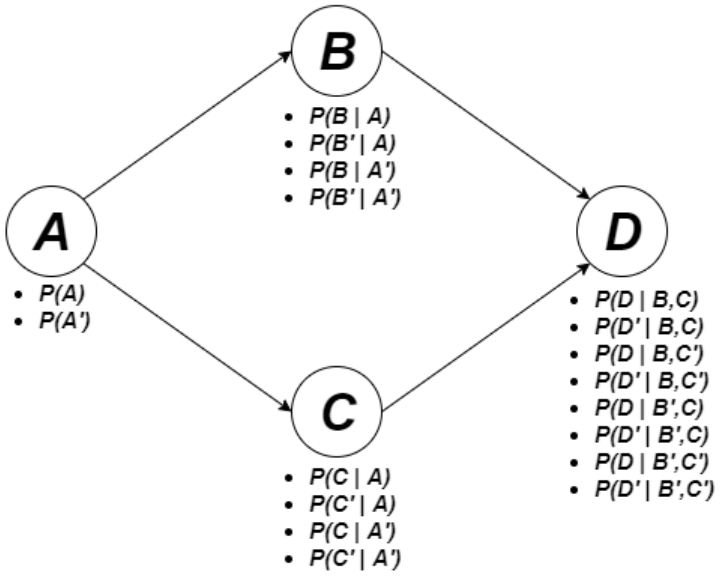

Figure 3.14: Bayesian network with CPTs

In the case of **root** nodes, the table is made out of marginal probabilities only. As the number of parents of a node increases, the number of probability cases increases exponentially, representing all the possible ways to describe the behaviour of the system.

Inference

The most useful characteristic of Bayesian models is the **belief updating** by computing the **inference** using evidence (Applying Bayes theorem, section 3.2.7). Conditional probability tables represent the probability of any event given its conditions (In this case, given the previous action of the attack process). These probabilities can usually be observed and calculated from parameters when working with discrete distributions, and can be used to calculate **hypothesis** and make diagnosis. Knowing how likely is an event to happen given all the possible previous events, it is possible to compute how likely a **previous step** would be to be a parent of that event. For the model in figure 3.14, applying **Bayes theorem** it is possible to find out how likely is ***A*** to be a parent of ***B*** by setting a prior value for ***A*** and adding it to the equation with the conditional probability of ***B*** given ***A***.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{3.34}$$

When applying the rule, **likelihood** and **prior** are divided by the **normalization constant**. The normalization constant of the son node is equal to the probability of that node to be in the indicated state. In this case, the normalization constant for $\boldsymbol{B}$ is the sum of all the joint probabilities of $\boldsymbol{B}$.

$$P(B) = \sum_i P\left(B, X_i\right) \Rightarrow P(B) = \sum_i P\left(B|X_i\right) P\left(X_i\right) \tag{3.35}$$

Inference is one of the most important informative parts of a Bayesian graphical model, but it needs a high computational performance in case of a large network. In this case, exact inference can be computed, but it is necessary to apply **inferential algorithms** to approximate the exact value when the number of states is too big.

Prior estimation

One of the elements used to calculate Bayesian inference is an estimation of the final result called **prior**. The value of a prior estimation can be very useful when there's some kind of knowledge available about the system and the possible outcome of the diagnosis, but there are different methods for the estimation of prior values depending on the situation. In this case, the main focus is on **noninformative priors** [59]. Noninformative prior estimation is used when there's no information about the possible outcome of the inference equation. There are many methods and algorithms for noninformative prior estimation, being **Jeffreys prior** [60] the most widely known distribution for one and multiple parameter cases.

Chapter 4

Case of study

The research and knowledge gathered in this project are the base for the design of a methodology for cyber attackers behaviour prediction using Bayesian networks. This proposal represents a tool included in the scope of a **cyber range**, which goal is to learn from data, provide understanding and represent a reliable method for cyber security training. This process is designed to be applied in a virtual environment, which represent a simulated network that is used to train and learn about cyber security processes. This virtual environment can be monitored, extracting data that can be processed in order to detect cyber attacks peformed inside the systems included in it. The dataset from which the described model is built contains information about detected cyber attackers actions inside a cyber range virtual environment.

4.1 Cyber range

The concept of cyber range refers to a virtual environment that is used to practice, learn and train for **cyber warfare**. This type of models usually represent a virtual system which emulates a real life infrastructure, providing tools and methods designed to understand and learn.

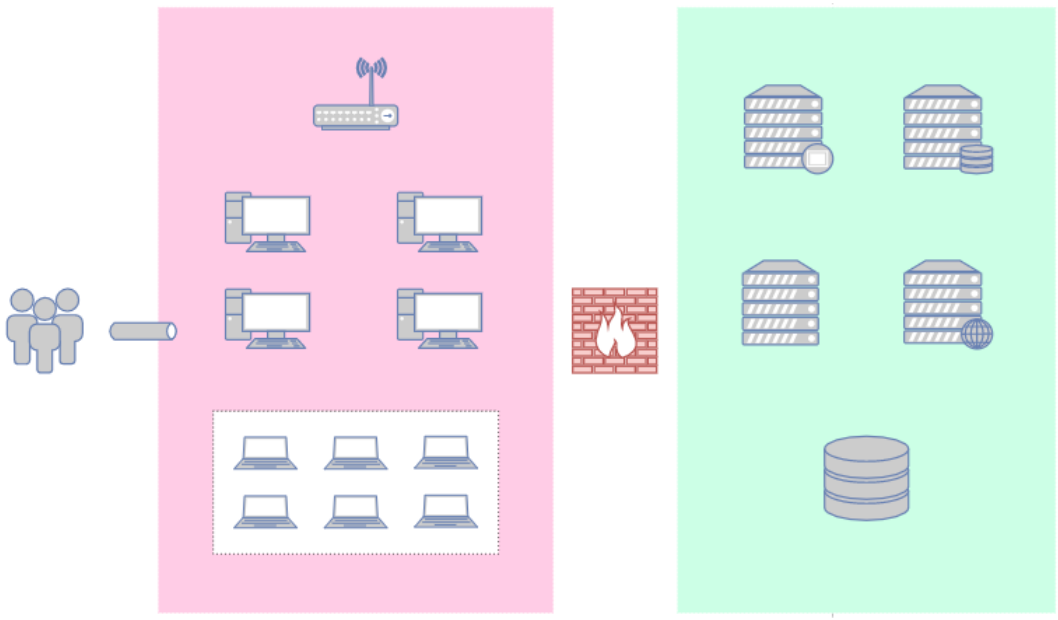

Figure 4.1: Example of a cyber range virtual environment

In this case, the proposed model represents a tool which application is intended to be made in a cyber range system to learn from data and predict cyber attacks. This specific system is used for the cyber security training of Computer Science students. More specifically, it is designed to be attacked and compromised by them, being able to find weaknesses and penetrate into the system. The goal of the method described in section 5 is to learn from the actions performed by the students, representing them as steps of attack processes, analysing the influence and relations of the actions and building patterns.

4.2 Assumptions

The described model is planned to work under some constraints that create a suitable environment to apply this method. These conditions define the properties of the dataset used to trace and build a behavioural prediction model.

Here are described the assumptions that make up the starting point for the application of the probabilistic prediction algorithm.

1. **Cyber attack information:** It is assumed that there's enough detectable information about the actions made by cyber attackers inside the network.

2. **Success:** Information about the success or failure of the attacks performed is available and possible to detect. In this case, in order to emulate a real case and apply a pattern searching method, it's assumed that an **unsuccessful** action can't be the previous step for another event.

3. **Identification of cyber attackers:** It is possible to identify the source of every action and classify all of them.

Taking these assumptions as a starting point to create a dataset, the information is analysed and modelled to create the graph, tracing patterns and performing the necessary calculations.

4.3 Data

In order to successfully trace and build patterns, a complete dataset of detected **cyber attacks** is necessary. In this case, a dataset that simulates the actions performed by 300 students inside a cyber range is analysed. The expected outcome after the application of this model to the dataset is a Bayesian Network that describes the typical behaviour of the students inside the system. The events contained in the simulated dataset define **six** possible complete patterns. It's important to understand that smaller parts of a complete pattern (graph) can also represent a pattern (Subgraph).

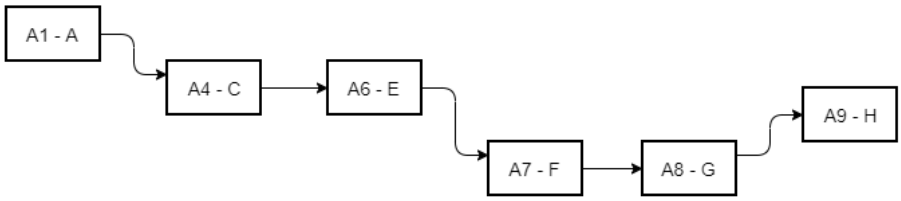

Figure 4.2: Possible pattern 1

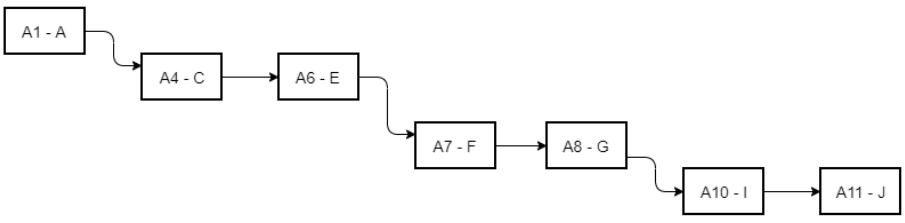

Figure 4.3: Possible pattern 2

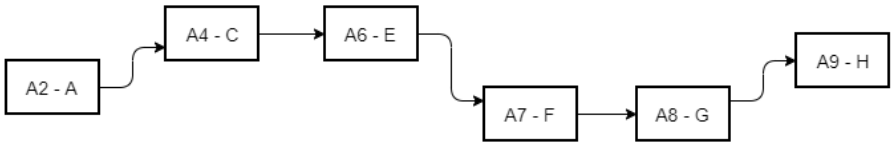

Figure 4.4: Possible pattern 3

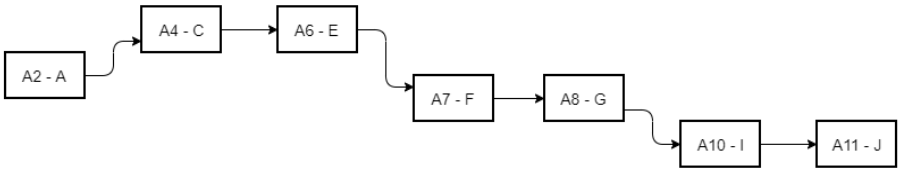

Figure 4.5: Possible pattern 4

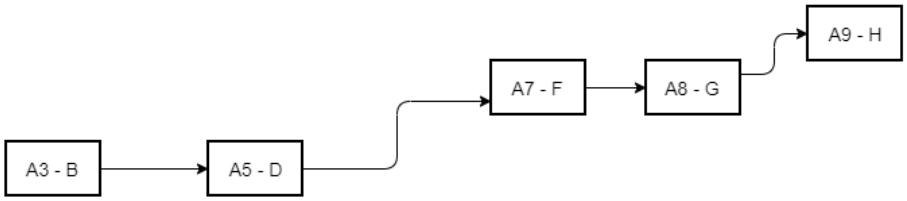

Figure 4.6: Possible pattern 5

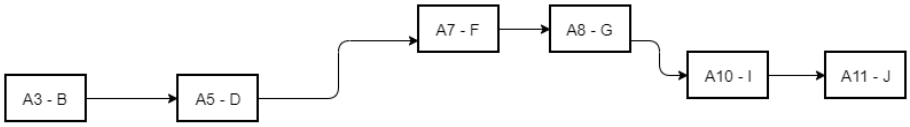

Figure 4.7: Possible pattern 6

All the possible patterns that can be found in the dataset compose the main configuration of the attack process that is simulated; in other words, the result of the combination of all the possible single patterns is the behavioural process simulated in the dataset.

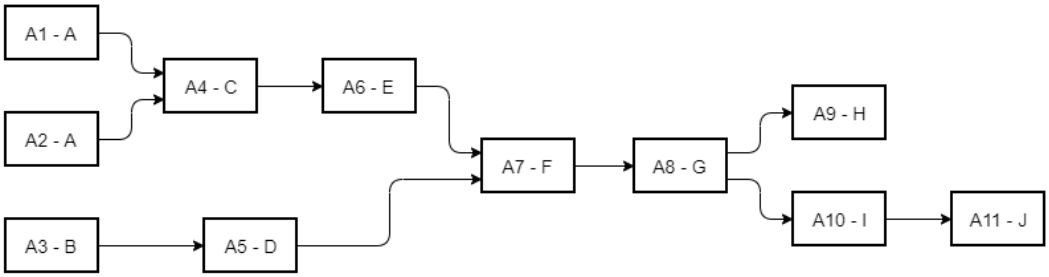

Figure 4.8: Simulated behavioural main scheme

The resulting dataset represents a list composed by detected cyber attacks. Every element of the list represents a detected event, and gathers information about the **type of attack**, the **name** of the attacked host, the **date** and **time** it was performed, and information about the **success** of the action. The elements of this list represent steps of a process, which means that the nature of the information is defined by the following properties:

- The next step of the process can only represent an event that is present after a successful performance of the previous action.
- An event that represents a child of a previous action is placed forward in time than its parent.

Following these conditions, the process to simulate a real dataset of cyber attacks performed by students has the following configurations:

1. The total amount of cyber attackers that can be found in the dataset is 300. Every attacker has an identifier assigned, which simulates an IP address.
$$10.0.0.\{0-299\} \tag{4.1}$$
2. The patterns followed by the attackers have three possible starting points (A1 - A, A2 - A, A3 - B). The identifiers are divided into this three first steps, starting and following the process from them.
$$\mathbf{A1-A} \Rightarrow 10.0.0.\{0-99\} \tag{4.2}$$
$$\mathbf{A2-A} \Rightarrow 10.0.0.\{100-199\} \tag{4.3}$$
$$\mathbf{A3-B} \Rightarrow 10.0.0.\{200-299\} \tag{4.4}$$

3. The date and time of the first actions are randomly generated for every attacker id. In case of a successful action, the next action of the process is created and associated to the same attacker, adding the necessary time difference between actions. This process simulates a real attack pattern and its progression in time. The time differences applied to every transition are shown in figure 4.9.

4. Every attack associated to an attacker identifier is created and represented combining the name of the attack and the targeted system. For instance, an attack **A - A1** represents an action **A** performed in the system **A1**.

5. Success values are added randomly as a binary variable (1 if success, 0 if fail). When an action is associated to an attacker id, this value is also set in the object. In case the success value is 0, it's considered that the attacker failed, and no more actions are associated to the same identifier.

6. In order to create the next events after the starting points, the whole set of actions is analysed, looking for the previous action (Attack and host) of the event that is being created, and the identifier associated with it. In case the success value of the previous action is 1, the next event is generated and associated with the same attacker id. This configuration simulates a real pattern, where the process of penetrating inside a network is divided into smaller steps in different systems.

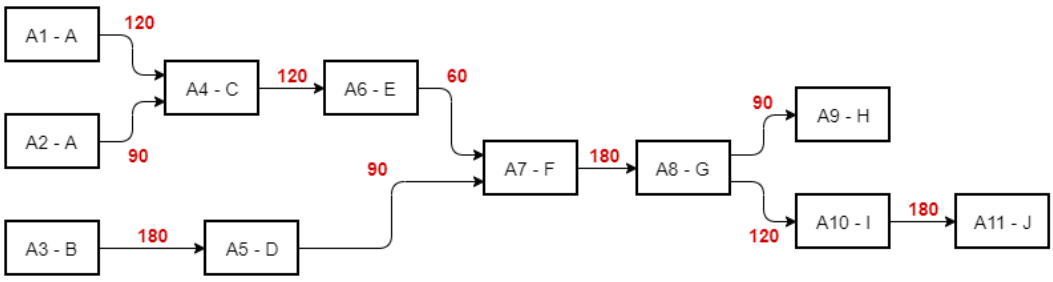

Figure 4.9: Time difference between steps (Minutes)

This simulation creates a list of events. Events represent actions performed by cyber attackers to a certain host and their success, which value is randomly assigned. The chains of events followed by every attacker can represent any possible pattern that a real student could follow inside a virtual system in this case. The result is a full dataset of detected cyber attacks than can be filtered and processed, representing the starting point of the algorithm.

4.4 Real case

The simulated dataset described in section 4.3 represents the main structure of an attack process. As it has been described, tracking the actions of cyber attackers and finding patterns requires the important task of establishing a **model**, defining what represents a **step** and what represents an **association** between them.

The dataset used to perform the simulation represents **actions** performed on different **systems**. It is important to understand what these actions are, what is their nature depending on the system they are performed in and what is their **purpose**. In this case, an action can be represented as a **cyber attack** or, depending on the dimension, a step of a greater process. These steps must be defined as part of a model in order to be able to detect them and use them as key parts of a pattern. Depending on their purpose or the target, cyber attacks can be classified, as it can be observed in the **MITRE ATT&CK** Matrix for Enterprises [61], which represents a catalogue where the typical threats and attacks performed in an enterprise environment are classified, depending on their nature or the goal followed by the actions.

In the case developed and simulated for this project, actions are represented as steps of a process, which represent cyber attacks performed in a certain system. The creation of a pattern is built based on a model of the typical behaviour of cyber attackers, where associations between steps represent a recursive sequence from one action to another. Understanding the nature of an attack and its purpose can be helpful in order to represent these associations and understand the results of this algorithm. This information becomes specially helpful when the understanding about specific threats of every system is taken into account, like **Initial access** or **Credential access** techniques for systems with a public interface (Web servers) or **Lateral movement** for applications or remote connection servers.

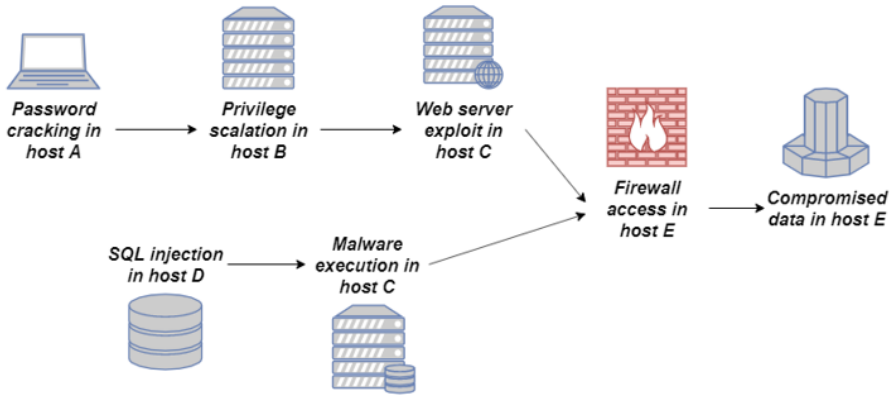

Figure 4.10: Real case simulation process

In order to understand the adaptability of this model, as well as analysing how efficient its performance can be in a real situation, an example of a real case based on real cyber attacks has been studied. This case (Figure 4.10) represents a simulated dataset of typical attacks. These attacks represent the steps of a whole process, which can be considered an attack pattern inside a cyber range:

- Password cracking (Brute Force) performed on personal device (Laptop)
- Privilege scalation performed on server
- Exploit execution on web server
- SQL injection attack performed on database
- Malware execution on storage server
- Firewall access achieved on core system

The proposed model contains the elements that can be usually found in a cyber range, which is the environment for which this project is developed, and represents the most typical model of network and resources access architecture for cyber security training.

It is important to understand that systems are constantly evolving, and the technological risk increases as digitalization does. Cyber attack techniques and hacking methods belong to a constantly innovating world, which makes it necessary to make systems safe, as it can be observed in new attacks and threats like **False Data Injection Attack** [62], or the result of investigations

regarding to the **Risk analysis of cyber-physical systems** [63]. The nature of cyber security science makes this project a versatile proposal, being able to be applied to any type of attack, event dataset or cyber security training environment, representing an effective method for any real scenario with the ability of measure and detection.

Chapter 5

Method

5.1 Introduction

All the research and investigation made in this project are focused on probabilistic analysis and modelling. The main goal of the work described here is to gather information about probabilistic models and methods to learn from data, provide understanding and use it to find patterns and predict the behaviour of cyber attackers. In this section are described the process of filtering, model structure construction and causality analysis, as well as the diagnosis process using **inferece** and the construction of the network, explaining the solutions and decisions taken in order to fulfil the main issues stated in section 1. The model described in this document is based on the information obtained from monitoring the systems of a virtual network, and the main goal is to create a model to represent patterns and attack processes based on Bayesian analysis, to find and represent information about events and dependencies between them, and analyse the nature of the typical actions made by cyber attackers inside the network. This model is meant to be scalable, giving the chance to improve it and be used to analyse any dataset that could provide enough information to find a pattern and calculate probabilities.

Figure 5.1: Attack list

As it was explained in the previous chapter, the dataset from which the model is built consists of a list of cyber attacks like the one in figure 5.1, which attributes and information are described in table 5.1.

Attack class	
Attacker	Identifier of the attacker
Name	Name of the action performed
Host	Name of the target
Datetime	Datetime information
Success	Information about success

Table 5.1: Attack class elements

In this case, every attack instance provides information about the responsible host (Attacker id), the type of attack detected, the targeted host, the date and time when it was performed, and the **success** of the action. This information is first filtered to construct discrete variables, and then analysed and processed in order to find dependencies, create patterns, trace them and calculate conditional probabilities and inference. The information obtained from the whole process, and from which the graphical model is built, is gathered in a **state** list.

State class	
Name	Name of the node
P. Success	Marginal probability of success
P. Failure	Marginal probability of failure
Parents	Parents of the node
CPT	Conditional probability table
Prediction	Table of calculated inference

Table 5.2: State class elements

This class contains the necessary information to build a Bayesian network. Every state represents a node of the network as a discrete variable with its marginal probabilities, also gathering the information of its dependency relations with other nodes as a **parent set**, the conditional probabilities observed and calculated from the dataset (**CPT**), which define its behaviour with regard to the other elements in the graph, and a diagnosis table including the predicted probability values as the result of the **inference** calculation process.

5.2 Space

The construction of Bayesian networks is based on probability spaces. In other words, the probability space of a model is the set of variables that represent the **nodes** of the network.

$$P = \{X_1, X_2, X_3, ..., X_i\} \tag{5.1}$$

The first step of this method is to define the probability space of the model, which includes the different events of the dataset. A dataset from which a pattern can be found and built includes common actions and repeated events, and every single event found in the dataset is added to the probability space of the model and represented by a node in the network, no matter the frequency of appearance. The main goal of this model is to describe the behaviour of cyber attackers inside a network, and that's why it's important to analyse and represent all the actions, event if they're isolated and can't be associated with an specific pattern.

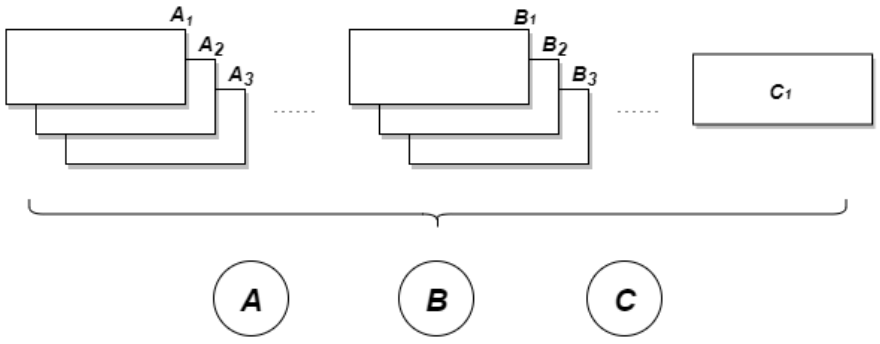

Figure 5.2: Nodes filtered from attack list

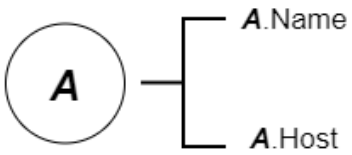

Figure 5.3: Node composition

Every detected attack performed inside an specific system represents a node, which is labelled combining the name of the performed attack with the name of the targeted host, and every node represents a variable of the probability space. In this case, variables are defined as **discrete binary variables**, which means that they have a **finite** number of samples and two possible values. With this definition, the nature and marginal probability of the events can be defined and computed by applying the basic rule of probability for binary discrete distributions.

$$A \begin{cases} Success & \text{if } p \\ Fail & \text{if } q = 1 - p \end{cases} \tag{5.2}$$

The number of samples of the variables represent all the times the same **attack** was detected in the same **host**, allowing to calculate the total number of successful and failed attempts.

$$p = \frac{\text{Number of success samples}}{\text{Total number samples}} \tag{5.3}$$

The result of this process is a probability space composed by the events detected in every different system included in the network, which represents the base from which a **Bayesian network** is built. Once the variables are defined, the next step is to set dependencies and build the structure of the model.

5.3 Structure

The elements that compose a Bayesian network are **nodes** and **edges**. Nodes represent the objects of the graphical model, and edges define the relations between them. As it was explained in section 3.1, this type of model is based on a **Directed Acyclic Graph**, which means that the connection between two nodes represents a directed **causal** dependency relation between them.

The problem of learning Bayesian networks can be divided into two different methodological approaches [64]:

- **Traditional approach:** Also known as **manual** approach, refers to the Bayesian networks structure construction by human experts and engineers, collecting data manually and representing the association between variables as **causal relations**, adding a directed edge from cause to effect. This method supposes a highly time-consuming task.

- **Learning approach:** Also known as **automatic learning** approach, is composed by the heuristic methods used to calculate Bayesian networks structures, which main approaches are described in section 3.3.6.

The classical construction of a Bayesian network structure from data is usually based on a probability space and a distribution of **conditional probabilities** of the variables, which are used to find dependencies between them. Applying heuristic **structure learning** methods it is possible to build a reliable graph to represent relations and fulfil the requirements of a Bayesian network model.

As it was described before, heuristic structure learning methods are divided into two main approaches, **score-based** and **constraint-based**. Score-based methods suppose an efficient way to build a model by comparing all the possible graph structures that fulfil the requirements of an specific case, using scoring functions like **Bayesian information criterion** [65], but although these methods are efficient and robust, they also suppose a high computational cost that makes them inefficient for this special case.

The nature of the probability space in this case, given the special environment it is based on, allows to apply some conditions for the construction of the structure, and makes it more suitable for a process included in the scope of a constraint-based approach [66]. In this case, applying a reasoning method to calculate the topology of a behavioural pattern also needs a proper application of the learning process for some specific constraints. The key of constraint-based methods is the usage of **independence tests**, which results represent the constraints. The performance of these algorithms is to compute the independence factor between two variables for a certain subset of values, representing totally independent variables $\boldsymbol{X}$ and $\boldsymbol{Y}$ for a subset of variables $\boldsymbol{S}$ as:

$$X \perp Y|S \tag{5.4}$$

The subset of variables for which an independence test is made is usually composed of those variables who are included in the **Markov blanket** of both

analysed variables, which decreases the number of calculations needed, and supposes that the computational cost of these algorithms, specially for high datasets, is normally lower than the score-based approach methods. Constraint-based methods are based on both the **Causal Markov condition**, which defines every variable as independent from its non-descendants, and the principle of **faithfulness** or **stability**, which means that it's not possible to represent a causal relation between two variables of which mutual independence has been computed and established [67].

Learning process

In this case, a learning process must be designed and applied in order to build a topology which represents cyber attackers behavioural patterns by learning from their actions. Applying a model like this for behavioural patterns means that cyber attacks are treated as objects that influence the existence of others, and act as **cause** or **effect**. This concept accurately defines the nature of cyber attacks as **steps** of a process for this case.

The nature of cyber attackers behaviour, combined with the assumptions made to describe the dataset, represent a set of conditions and constraints that can be used to test the dependency between the filtered nodes:

- **Chronological order:** The behaviour of cyber attackers is represented by a series of actions ordered in time. Other constraint-based solutions that don't depend on the dataset order [68] can be used, but it is an strong and important condition that must be taken in account in this case.
- **Identity:** The source of an action can be identified.
- **Success:** An object that doesn't represent a root node is considered an effect of a previous successful attack.

For this special case, performing independency tests for every pair of variables requires a higher effort due to lack of knowledge about the association between nodes at this point, and that's why this test is made searching for direct **dependencies**. The first step is to divide the dataset into a **samples matrix**, separating the actions associated with every different attacker id.

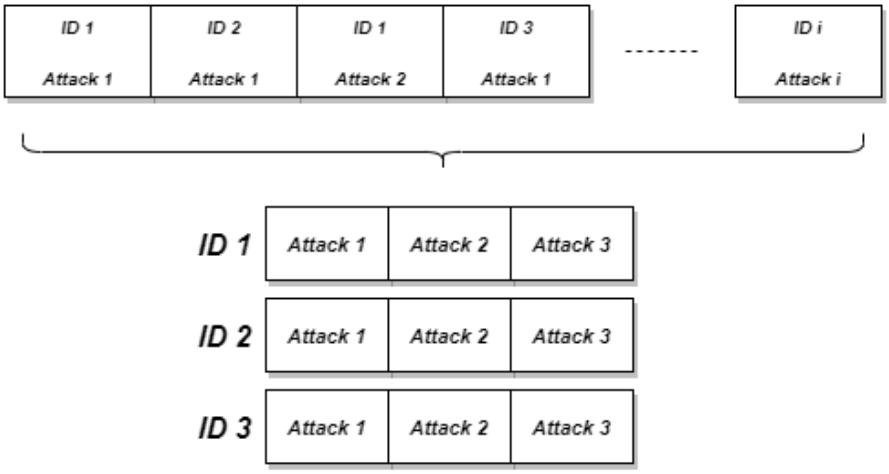

Figure 5.4: Attacker actions matrix

The result of this filtering process is a matrix in which every row is a **chronologically ordered** list of the actions performed by an attacker. The next step is to use this resulting matrix in order to find dependencies between nodes, which lead to the construction of a pattern. It's important to take in account the **causality** relation property between dependent nodes in this case, from which only a successful attack can be the cause of another one. Due to this condition, the resultant matrix is filtered in order to remove unsuccessful actions.

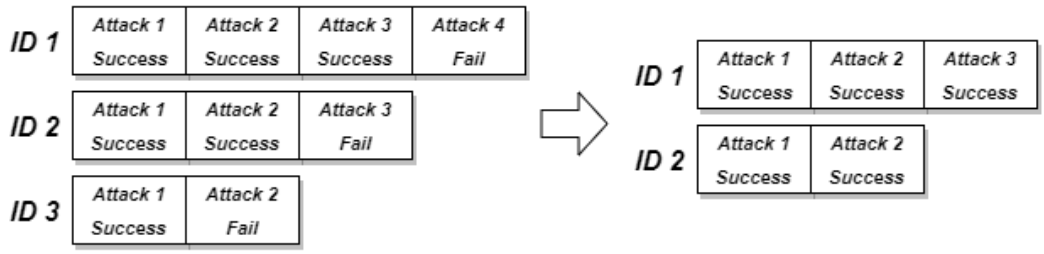

Figure 5.5: Matrix filtering example

The outcome of this process is a list with all the series of successful actions followed by the different cyber attackers inside the system. At this point, it is possible to analyse which events represent a previous action, or **cause**, for every node; in other words, it is possible to calculate the **parent set** for all the variables in the probability space. The process applied to every node of the probability space for its parent set search is:

1. Search for the series in the matrix where the attack is present, ignoring the series where its presence is not found.

2. For every appearance of the attack in a series, the first previous action is analysed (In case it exists), and its presence as a previous step of the attack is calculated, comparing it with the rest of the appearances of the attack. This process is made for all the different first previous actions that are found in the series, for every appearance of the attack.

3. Depending on how often a previous action is present before the analysed attack, it's considered a parent of it or not. It is important to avoid false patterns, and that's why a process performed only one time can't represent a typical behavioural process of cyber attackers. When an action is found to be a previous step of the attack three times or more, it can be considered a cause of the attack and a dependency relation between them is established, including it in the parent set of the attack. This threshold for dependencies establishment could also be computed based on probabilities, but it could lead to missing relations for small datasets, which is the case of this specific network.

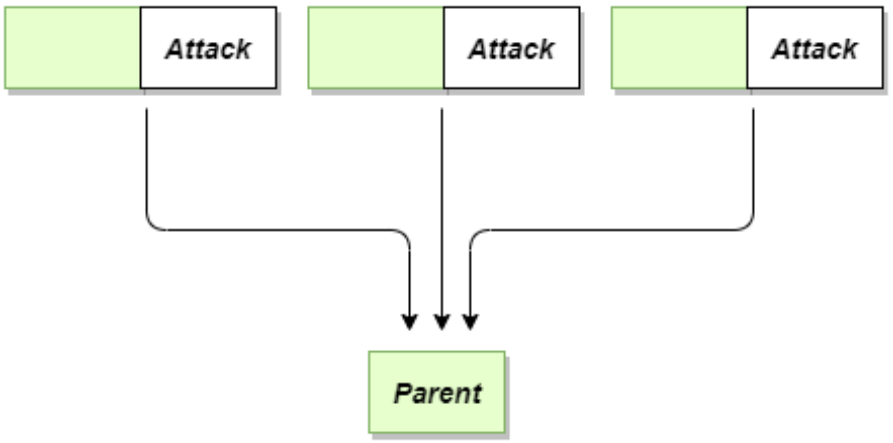

Figure 5.6: Previous action as a parent of the attack

After applying this algorithm, all the dependency relations between the elements of the probability space have been set, and every state has now a parent set. Graphically, the causal relation between every state and those in its parent set is represented with an edge, directed towards the analysed state.

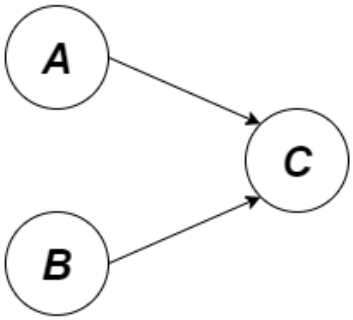

Figure 5.7: Example of an state with two parents

The parent set of the example in figure 5.7 can be mathematically described as follows:

$$\text{par}(C) = \{A, B\} \tag{5.5}$$

Due to the natural properties of this case, a process associated with the traditional approach of Bayesian networks structure learning is used in order to find causal relations, which also leads to an empirical **independency test** process. The parent set information of every state allows to construct their **Markov blanket**, considering every variable **d-separated** and, indeed, **independent** from any other variable that has no presence in its parent set and neither includes it in theirs; in other words, a variable ***A*** is considered independent from ***B*** if they are not a parent or a son of each other in any case.

5.4 Conditional probability tables

The construction of a Bayesian network from discrete variables is based on a probability space composed of discrete variables and a **probability distribution**. This probabilistic graphical model represents causal relations between variables using conditional probabilities, and make it possible to understand how likely all the possible situations are to happen. In this case, all the knowledge is achieved by learning from a dataset of detected cyber attacks, and it's possible to observe and calculate conditional probabilities by analysing the actions and the dependencies that have been already set. Every node in the model has a **Conditional Probability table** associated that describe the probability of the represented event to happen given all the possible situations where it is involved. Given a set of parents, the CPT is calculated and adapted to the nature of the state that is being analysed. Depending on the situation, the size

of the CPT for discrete binary variables increases exponentially as the number of parents do. For **root** nodes, all the possibilities are defined only by their marginal probabilities.

Figure 5.8: Single node

CPT
P(A)
P(A')

Table 5.3: CPT of a single node

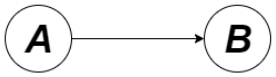

Figure 5.9: Two nodes connected

CPT
P(B|A)
P(B|A')
P(B'|A)
P(B'|A')

Table 5.4: CPT of a single-parent node

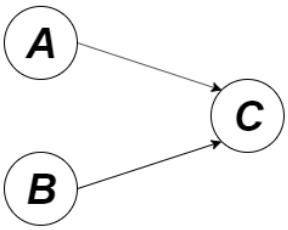

Figure 5.10: Two nodes converging in one

CPT
P(C|A,B)
P(C'|A,B)
P(C|A,B')
P(C'|A,B')
P(C|A',B)
P(C'|A',B)
P(C|A',B')
P(C'|A',B')

Table 5.5: CPT of a multiple-parent node

Conditional probability information is computed by analysing the **evidence** found in attack processes. Given all the possible scenarios of previous conditions, samples are filtered and the behaviour can be observed from filtered

information. All the possible outcomes (Given the previously found dependencies) are covered and have a measured probability.

For this process, the matrix of series is not filtered, due to the necessity of including the development in case of failures. Every variable has two possible outcomes, and it's possible to compute the probability of both different results for every possible situation by applying the basic probability concept represented in the equation 5.3. For instance, given a variable ***A*** with a single parent ***B***, the probability of success of ***A*** given ***B'*** is the result of dividing the number of samples where ***A*** is successful after a failed attempt of ***B*** by the total number of samples where ***A*** is successful.

$$P(A|B') = \frac{\text{Times variable A is successful after B failed}}{\text{Times A is successful}} \tag{5.6}$$

Given a parent set, all the possible conditions are represented as the number of possible binary combinations of a number of bits equal to the lenght of the set, applying them to filter the series matrix and calculate probabilities.

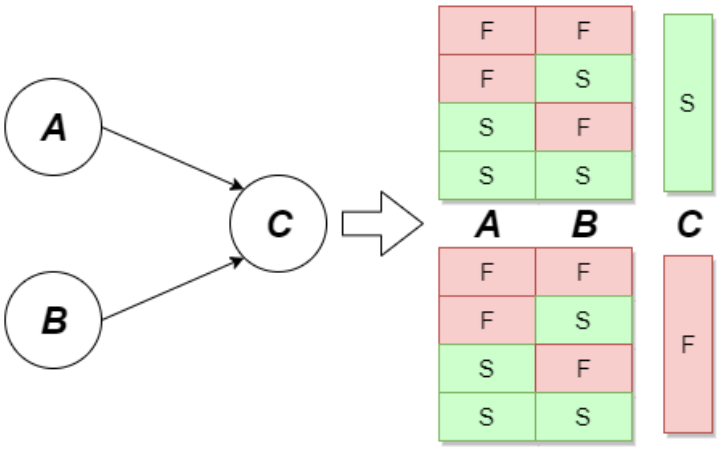

Figure 5.11: Binary combinations for a parent set with two elements

In this case, the scope for every variable is limited by the attacks included in its parent set, and its CPT contains the conditional probability for all possible outcomes (Success or Fail), given all the possible values of its parents. The condition of discrete binary variables makes the number of entries in the CPT to double its quantity every time a new element is included in the parent set.

The result of this process is a **Conditional Probability table** associated to every state. The values included in the CPTs define the relations between nodes, and represent the level of influence between cyber attacks in a be-

havioural pattern, which is a very useful property for a cyber attackers behaviour tracing system. This information is one of the most important parts of a Bayesian network, and also supposes a key factor to compute **inference** values.

5.5 Inference

When all the possible states are known and their conditional probabilities are calculated, **Bayesian inference** is computed by applying **Bayes theorem** for every possible situation. At this point of the process, the elements that compose the DAG are already defined like discrete binary variables and their marginal probabilities, and their dependencies and relations can be analysed and represented using the information of their parent sets and CPTs. Using this information, it's possible to measure and represent **predictions** in order to construct a Bayesian model. Calculating the inference is the process to determine and represent how likely is an event to be a **previous step** for other attacks, or the probability of a state to be a parent of the next one.

The **posterior** probability for every state given the calculated evidence (CPTs) is computed by applying Bayes rule, translating the present condition to the different parts of the equation.

1. **Likelihood:** The likelihood factor is the evidence given to the equation. It is represented by the observed conditional probability given the situation that is being calculated. So, to compute the conditional probability of an even ***A*** to be a parent of an event ***B***, the likelihood factor is the conditional probability of ***B*** given ***A***.

$$P(A|B) \Rightarrow \text{ Likelihood factor: } P(B|A) \tag{5.7}$$

   In case of more than one parent, the likelihood factor is computed taking in account all the possibilities that fulfil the conditions.

$$P(A|B) \Rightarrow \quad \text{Likelihood factor: } P(B|A,C) = \prod_i P\left(B|A,C_i\right) \tag{5.8}$$

2. **Prior:** The prior factor is the expected outcome of the inference. In this case, the only available information about the analysed event are its marginal and conditional probabilities. Given the situation that states are **binary discrete variables** and there is no available information about the expected outcome, the marginal probability of a variable can be used as a prior value.

3. **Normalization constant:** The likelihood and prior factors are divided by the normalization constant. This factor provides information about how likely the **son node** is to be in the indicated state; in other words, it represents the isolated value of the variable in a certain state. The normalization constant isolates the probability of a value by combining conditional and marginal probabilities of the variables that influence its behaviour.

$$P(B) = \sum_i P\left(B|A_i\right) P\left(A_i\right) \tag{5.9}$$

In case of more than one son, the normalization constant is computed as the **joint probability** of all the sons. It is usual that the sons of a node have no dependencies between them (The model has no cycles), so it's important to observe the possible conditional independency when calculating this factor.

$$P(B,C) = P(C)P(B|C) \tag{5.10}$$

$$\text{Conditional Independence: } P(B,C) = P(B)P(C) \tag{5.11}$$

The process of inference calculation in this algorithm is applied for every state of the model. The first step of this algorithm is to find the possible actions an attacker can perform after the analysed variable, which is made by analysing the presence of the variable in the rest of parent sets of the states list.

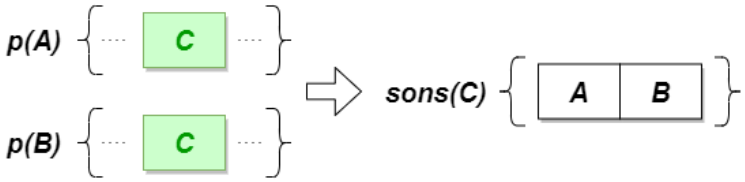

Figure 5.12: Sons association searching

The number of associations found in this process defines the number of entries of the resulting prediction table, being represented as all the binary combinations for a number of bits equal to the number of sons of the variable.

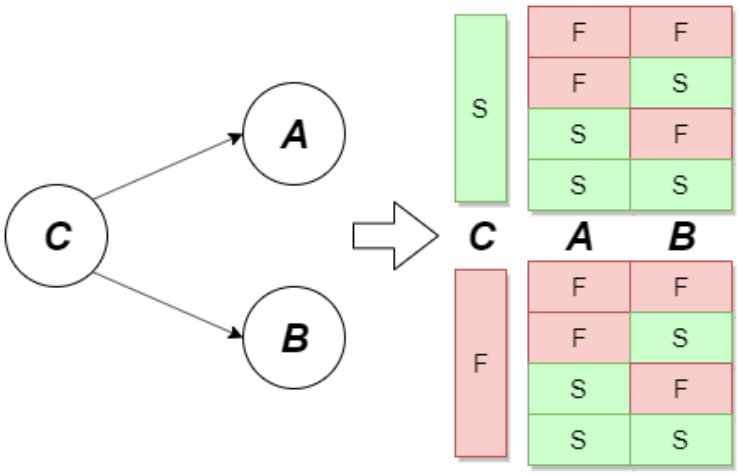

Figure 5.13: Binary combinations for a node with two sons

After defining the combinations, the previous theoretical concepts are applied to calculate the **Bayes rule** factors. These factors are computed using the information gathered in the states list. For instance, the likelihood of an state ***B'*** which is included in the parent set of an event ***A*** is equal to the conditional probability of ***A*** given ***B'***.

$$P(A|B') \tag{5.12}$$

In this case, the normalization constant represents the probability of ***A***, which represents the variable ***A*** in the state of **success**.

$$P(A) = P(A|B)P(B) + P(A|B')P(B') \tag{5.13}$$

Given the likelihood and normalization constant, and taking the marginal probability of the variable ***B*** in state of failure ***B'***, the inference model in this case can be built and computed.

$$P(B'|A) = \frac{P(A|B')P(B')}{P(A)} \tag{5.14}$$

The same process is applied for every state in the list, analysing their presence in the rest of parent sets of the list (Can't be included in their own parent set, no cycles are allowed) and calculating the inference for all the possible values and conditions. This process applies the same property of independence of a **naive Bayes classifier**. The results for every connected state are gathered forming a **diagnosis table**.

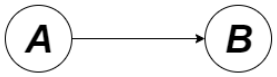

Figure 5.14: Two nodes connected

Diagnosis table
P(A|B)
P(A|B')
P(A'|B)
P(A'|B')

Table 5.6: Diagnosis table of a node with a single son

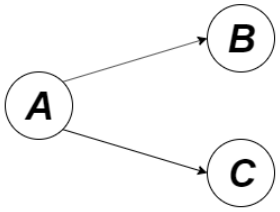

Figure 5.15: A node with two sons

Diagnosis table
P(A|B,C)
P(A'|B,C)
P(A|B,C')
P(A'|B,C')
P(A|B',C)
P(A'|B',C)
P(A|B',C')
P(A'|B',C')

Table 5.7: Diagnosis table of a node with multiple sons

After this process, a table with predictive probabilities is added to every connected state, completing the Bayesian networks model reasoning algorithm. At this point, all the elements of which the network is composed are defined, and the model is built as a directed acyclic graph designed to represent cyber attackers behavioural patterns and provide as much understanding as possible at every point. When different and conditionally independent patterns are found and there are no connected states between them, the graph is represented as a **poly-tree** structure which allows to observe different patterns in the same model.

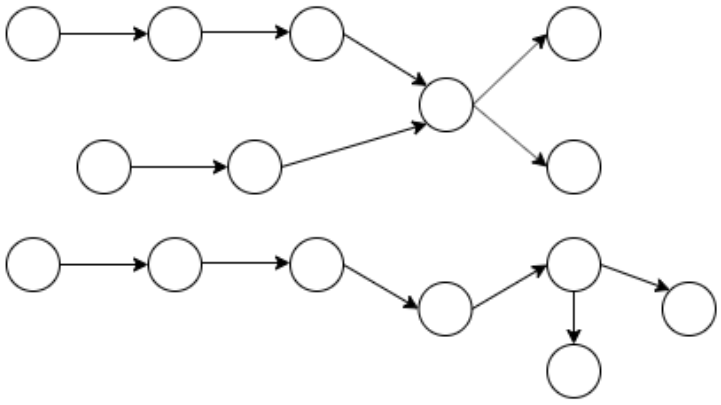

Figure 5.16: Poly-tree structure graph

5.6 Model construction

Once the previous processes are finished, the resulting state list can be used to build the model. The first step of this process is to generate the elements of the graph. This step is divided into two operations: Generating a **node** for every state, adding the name of the state as a label, and generating **directed edges** from every element found in their parent sets.

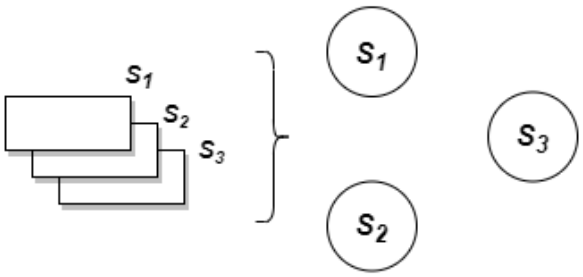

Figure 5.17: Nodes generated from states list

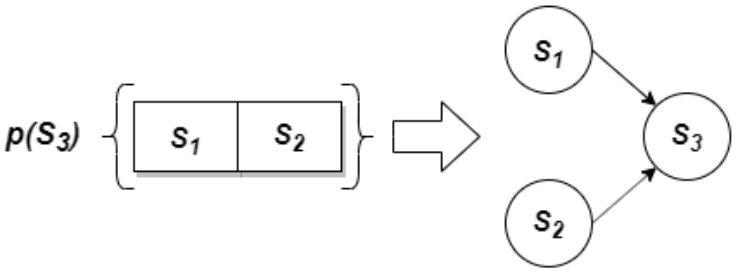

Figure 5.18: Edges generated from parent set

Once the elements of the graph are generated, the next step is to build the graph and represent the processed information.

5.6.1 Application

The code needed to apply the filtering processes and mathematical methods described before represents the application model, which process the obtained data in order to build the resulting Bayesian network. All the mathematical processes designed for this application, as well as the module for graphical representation, are implemented using **Python**, which is considered the proper platform for the design of this model due to its good performance for probabilistic data analysis and representation.

Figure 5.19: Python logo

5.6.2 Methodology

The application core is divided into **three blocks**, separating the filtering code from the mathematical algorithms and graphical representation.

1. **BN.py:** This is the main script used for the creation of the model. In this script is implemented all the process to filter data and find the parameters of the model, as well as the **graphical representation** method.

This code applies the method described before in order to find states and dependencies from the simulated dataset of detected cyber attacks, importing the functions included in **AnalysisModules.py** to calculate probabilities and create a Bayesian network. The graphical model is represented as a web application using **Dash**, creating a cyber security tracing system as an interactive graphical model that allows the user to explore all the available information about the parameters involved. (Appendix A)

2. **AnalysisModules.py:** In this module, all the necessary functions to generate simulated data and calculate probabilities are implemented. This script is imported as a function set for the main script and is applied to the filtered data. The content of this file is composed by the necessary functions to calculate **marginal probabilities**, the algorithm to **find dependencies** given a list of actions made by cyber attackers, the operations to calculate **conditional probability tables**, as well as to find **prior** values, **likelihood**, and compute the **normalization constant** to apply the Bayes theorem and calculate the inference given a set of states. This block also contains operations to find **root** nodes and remove **cycles** to make sure the outcome meets all the requirements to be a Directed Acyclic Graph. (Appendix B)

3. **Style.css:** Dash is a python library that allows to create web modules using **HTML** in Python. This script works as an external stylesheet for the graphical application implemented in the main script. (Appendix C)

5.6.3 Graphical model

The result of the process applied to the simulated dataset described in chapter 4 is an accurate graphical representation like the one in figure 5.20. The observable similarity between the resulting graph and the pattern scheme of figure 4.8 shows the high level of accuracy of this method given the conditions of this case.

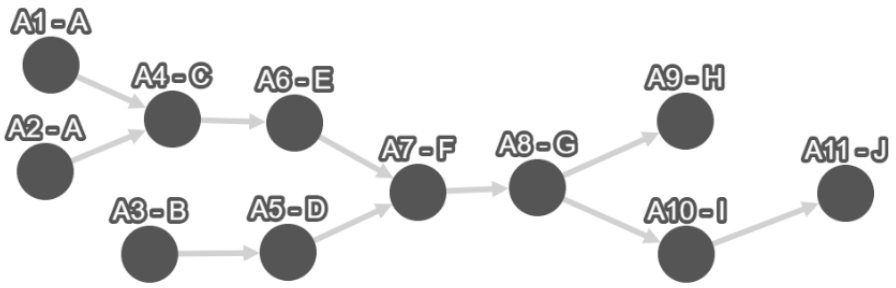

Figure 5.20: Processed graphical result

The resulting graphical model is represented as an interactive tool to analyse all the different points of the processed pattern. This means that the processed information included in the state can be shown when selecting a node, as well as the dependency relation between parent and son when selecting an edge. After the selection, the **CPT** and **diagnosis table** of the state are shown, providing the information about conditional probabilities and computed inference.

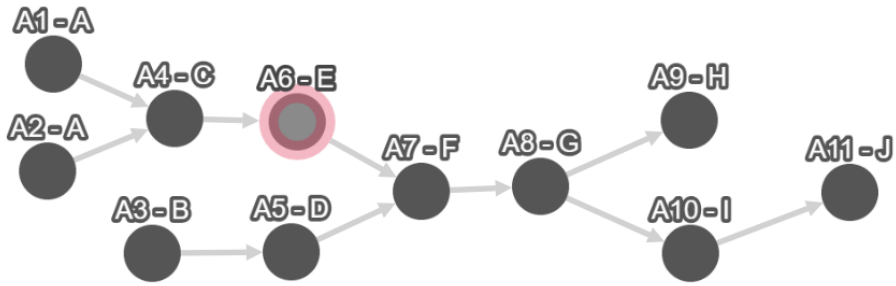

Figure 5.21: Processed model: Node selected

Diagnosis table for "A6 - E"

- P(A6 - E^|A7 - F^):	
- P(A6 - E^|A7 - F):	
- P(A6 - E|A7 - F^):	0.288
- P(A6 - E|A7 - F):	0.269

Conditional probability table for "A6 - E"

- P(A6 - E^|A4 - C^):	
- P(A6 - E|A4 - C^):	
- P(A6 - E^|A4 - C):	0.45
- P(A6 - E|A4 - C):	0.55

Figure 5.22: Information tables for the selected node

In the information tables, a blank field means **probability zero**. The accuracy of the probabilistic computation can be demonstrated by analysing the information tables. In case of a node in a **serial connection** as a single parent, the inference results confirm the nature of the simulated data (The probability of an unsuccessful event to be a previous step of another event is equal to zero). This property is confirmed by the **inference values**, which are equal to zero for the cases where the previous step failed, and the CPT, which also represents all these cases as probability zero.

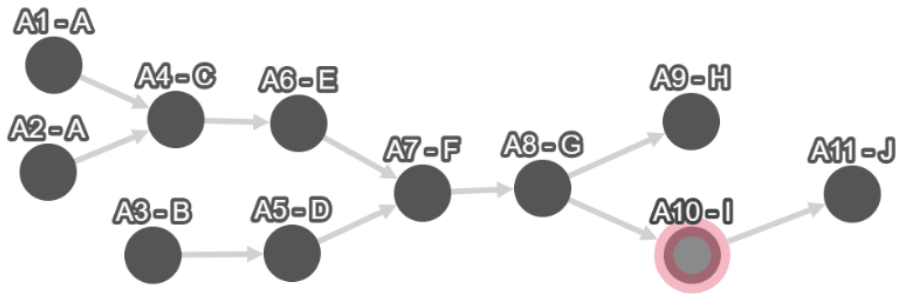

Figure 5.23: Serial connection node selected

Diagnosis table for "A10 - I"

- P(A10 - I^|A11 - J^):	
- P(A10 - I^|A11 - J):	
- P(A10 - I|A11 - J^):	1
- P(A10 - I|A11 - J):	1

Conditional probability table for "A10 - I"

- P(A10 - I^|A8 - G^):	
- P(A10 - I|A8 - G^):	
- P(A10 - I^|A8 - G):	0.429
- P(A10 - I|A8 - G):	0.571

Figure 5.24: Information tables for a serial connected node

In case of an state with more than one parent, Conditional Probability tables increase their size exponentially. This effect, previously explained in this chapter, is due to the condition of **discrete binary variables**.

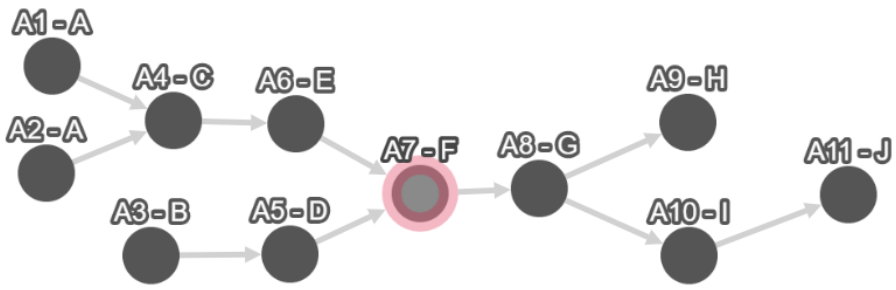

Figure 5.25: Node with more than one parent selected

Diagnosis table for "A7 - F"

- P(A7 - F^|A8 - G^):	
- P(A7 - F^|A8 - G):	
- P(A7 - F|A8 - G^):	1
- P(A7 - F|A8 - G):	1

Conditional probability table for "A7 - F"

- P(A7 - F^|A6 - E^,A5 - D^):	
- P(A7 - F|A6 - E^,A5 - D^):	
- P(A7 - F^|A6 - E^,A5 - D):	0.424
- P(A7 - F|A6 - E^,A5 - D):	0.576
- P(A7 - F^|A6 - E,A5 - D^):	0.455
- P(A7 - F|A6 - E,A5 - D^):	0.545
- P(A7 - F^|A6 - E,A5 - D):	0.436
- P(A7 - F|A6 - E,A5 - D):	0.564

Figure 5.26: Information tables for a node with more than one parent

In this model, root nodes represent the starting events of the analysed patterns. These nodes observable probability is limited to their marginal probabilities, which means they are not influenced by other states in any case.

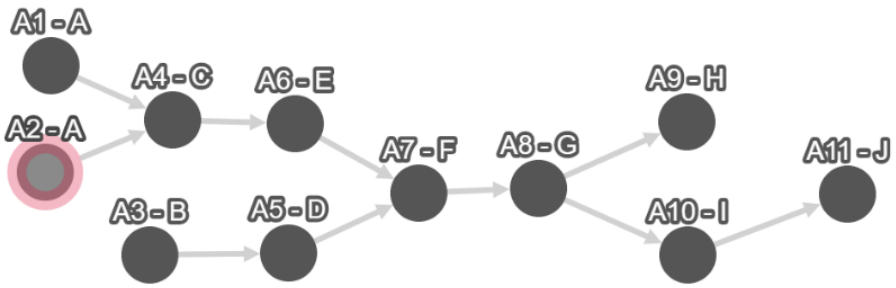

Figure 5.27: Root node selected

Diagnosis table for "A2 - A"

- P(A2 - A^|A4 - C^):	
- P(A2 - A^|A4 - C):	
- P(A2 - A|A4 - C^):	0.347
- P(A2 - A|A4 - C):	0.412

Conditional probability table for "A2 - A"

- P(A2 - A^):	0.51
- P(A2 - A):	0.49

Figure 5.28: Information tables for a root node

In case of leaf nodes, the appreciated effect is the opposite of the case for root nodes. A **leaf node** is considered the last event of the process, which mean that it doesn't represent a cause for any other event and, indeed, is not included in any parent set. For this type of nodes, there's no possible inference computation.

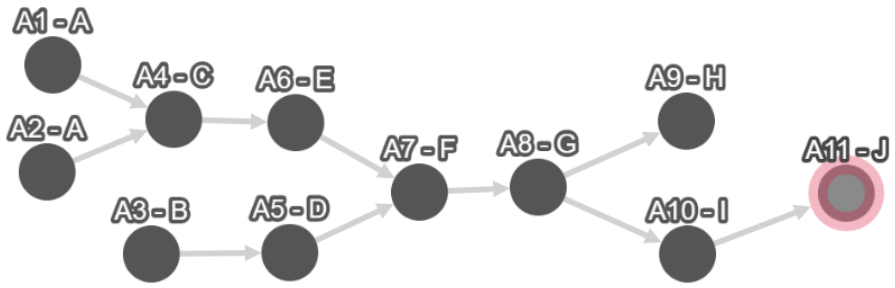

Figure 5.29: Leaf node selected

Diagnosis table for "A11 - J"

No information available.

Conditional probability table for "A11 - J"

- P(A11 - J^|A10 - I^):	
- P(A11 - J|A10 - I^):	
- P(A11 - J^|A10 - I):	0.25
- P(A11 - J|A10 - I):	0.75

Figure 5.30: Information tables for a leaf node

Directed relations can also be analysed, providing information about the parental relation they represent, as well as the **most likely outcome** of the relation.

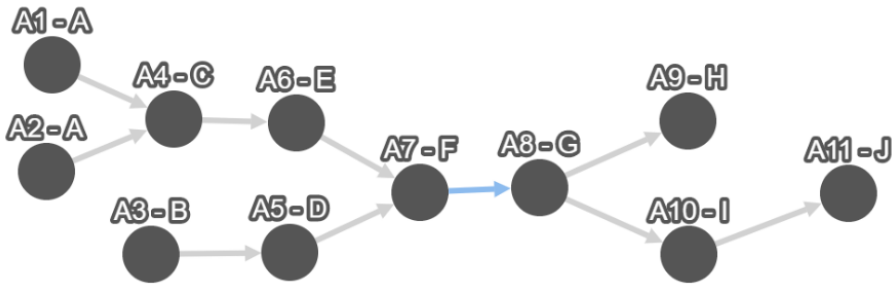

Figure 5.31: Edge selected

Found dependency between "A7 - F" and "A8 - G"

- P(A8 - G^|A7 - F):	0.548

Figure 5.32: Edge dependency information

Sometimes, specially when working with large datasets, an small probability to establish a **false dependency** is present. Usually, a false relation introduces a low probability connection in the model, which can create a cycle inside the graph, and it's easy to interpret and understand as an unrealistic connection. This can happen in a small number of cases due to the condition of **chronological order** of the simulated data.

In cases where there's not enough evidence to establish a dependency relation, nodes are represented as isolated events. In this special case, due to the random success value assigned to the simulated attacks, this can sometimes happen for the last steps of the process. Either way, this represents a minor problem which is almost not likely to be present in case of a dataset of real attacks.

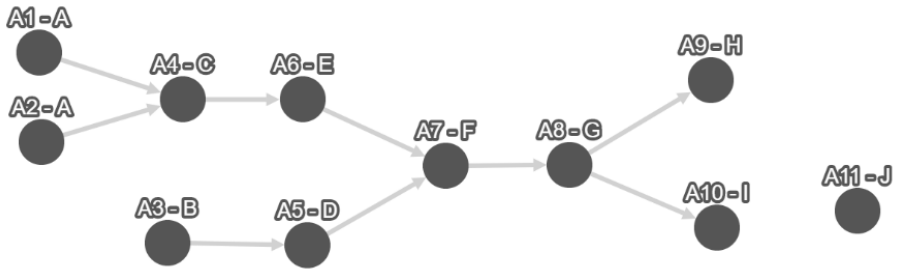

Figure 5.33: Unconnected node

In the case of the example described in section 4.4, the resulting network represents an easier way to understand the application of this process for a real scenario, establishing a reference of the accuracy and possibilites of the algorithm.

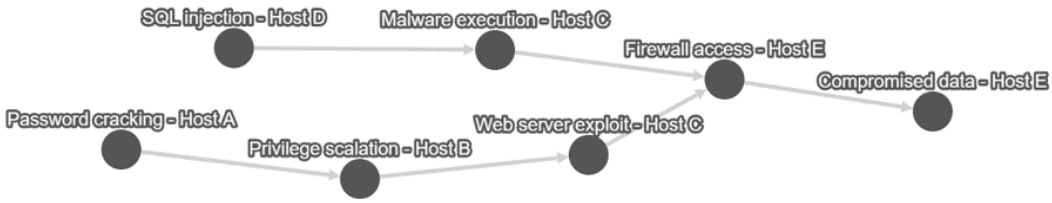

Figure 5.34: Processed graphical results for a real case

In this case, the information for every step can also be represented and analysed, having a reference of the probabilities at every step of the process.

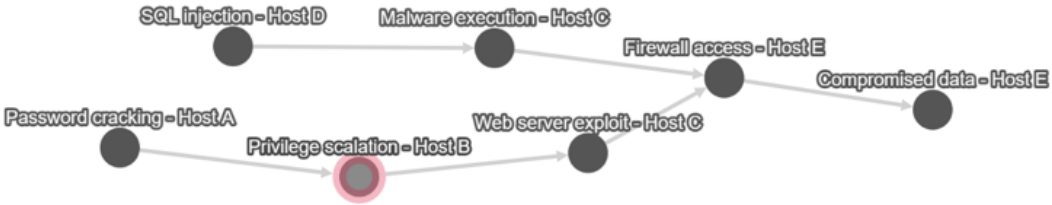

Figure 5.35: Real case - Node selected

Diagnosis table for "Privilege scalation - Host B"

- P(Privilege scalation - Host B^|Web server exploit - Host C^):	
- P(Privilege scalation - Host B^|Web server exploit - Host C):	
- P(Privilege scalation - Host B|Web server exploit - Host C^):	1
- P(Privilege scalation - Host B|Web server exploit - Host C):	1

Conditional probability table for "Privilege scalation - Host B"

- P(Privilege scalation - Host B^|Password cracking - Host A^):	
- P(Privilege scalation - Host B|Password cracking - Host A^):	
- P(Privilege scalation - Host B^|Password cracking - Host A):	0.458
- P(Privilege scalation - Host B|Password cracking - Host A):	0.542

Figure 5.36: Real case - Information tables of a single parent and son node

It can be observed that, following the same assumptions as the first simulation, diagnosis probability results are absolute for nodes with a single parent.

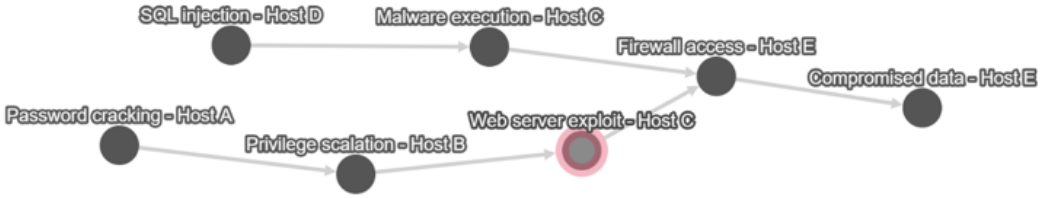

Figure 5.37: Real case - Node with a shared son selected

In the case of Figure 5.37, diagnosis probability computation results are also different for an event that represents a shared possible previous step for a single outcome.

Diagnosis table for "Web server exploit - Host C"

- P(Web server exploit - Host C^|Firewall access - Host E^):	
- P(Web server exploit - Host C^|Firewall access - Host E):	
- P(Web server exploit - Host C|Firewall access - Host E^):	0.343
- P(Web server exploit - Host C|Firewall access - Host E):	0.482

Conditional probability table for "Web server exploit - Host C"

- P(Web server exploit - Host C^|Privilege scalation - Host B^):	
- P(Web server exploit - Host C|Privilege scalation - Host B^):	
- P(Web server exploit - Host C^|Privilege scalation - Host B):	0.538
- P(Web server exploit - Host C|Privilege scalation - Host B):	0.462

Figure 5.38: Real case - Information tables of a node with a shared son

In the case of events where more than one action represents a possible previous step, it can be observed that the Conditional Probability table size increases exponentially.

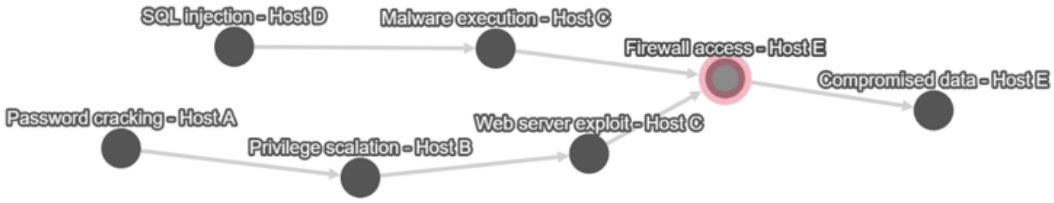

Figure 5.39: Real case - Node with two parents selected

Conditional probability table for "Firewall access - Host E"

- P(Firewall access - Host E^|Web server exploit - Host C^,Malware execution - Host C^):	
- P(Firewall access - Host E|Web server exploit - Host C^,Malware execution - Host C^):	
- P(Firewall access - Host E^|Web server exploit - Host C^,Malware execution - Host C):	0.5
- P(Firewall access - Host E|Web server exploit - Host C^,Malware execution - Host C):	0.5
- P(Firewall access - Host E^|Web server exploit - Host C,Malware execution - Host C^):	0.333
- P(Firewall access - Host E|Web server exploit - Host C,Malware execution - Host C^):	0.667
- P(Firewall access - Host E^|Web server exploit - Host C,Malware execution - Host C):	0.438
- P(Firewall access - Host E|Web server exploit - Host C,Malware execution - Host C):	0.562

Figure 5.40: Real case - Information tables of a node with two parents

In a real cyber attacker action pattern detected inside a cyber range, the resulting effects can be similar to the ones represented by this simulation. The actions that represent the initial steps of the process are represented by their marginal probability. As it can be observed in the simumlation, the opposite result can be found in the case of the last steps of this process, where the diagnosis probability can't be computed. This results can also be analysed in this case.

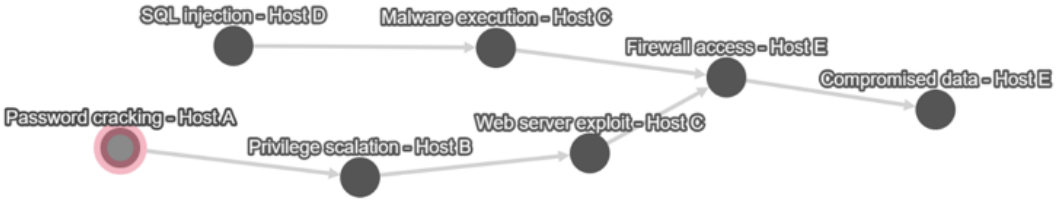

Figure 5.41: Real case - Root node selected

Diagnosis table for "Password cracking - Host A"

- P(Password cracking - Host A^|Privilege scalation - Host B^):	
- P(Password cracking - Host A^|Privilege scalation - Host B):	
- P(Password cracking - Host A|Privilege scalation - Host B^):	1
- P(Password cracking - Host A|Privilege scalation - Host B):	1

Conditional probability table for "Password cracking - Host A"

- P(Password cracking - Host A^):	0.52
- P(Password cracking - Host A):	0.48

Figure 5.42: Real case - Information tables of a root node

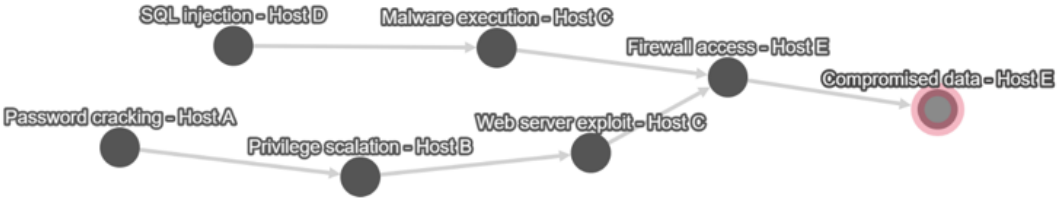

Figure 5.43: Real case - Leaf node selected

Diagnosis table for "Compromised data - Host E"

No information available.

Conditional probability table for "Compromised data - Host E"

- P(Compromised data - Host E^|Firewall access - Host E^):	
- P(Compromised data - Host E|Firewall access - Host E^):	
- P(Compromised data - Host E^|Firewall access - Host E):	0.667
- P(Compromised data - Host E|Firewall access - Host E):	0.333

Figure 5.44: Real case - Information tables of a leaf node

The final result of this process is the graphical representation of the cyber attacks performed by students in a cyber range. These actions are turned into discrete variables and represented in a Bayesian network, defining their properties and their influence with regard to the rest of elements in the probability space. The represented relations define the behaviour of cyber attackers, allowing to analyse the possibilities and most likely outcomes at every point.

The resulting model, compared with the pattern used to simulate the dataset, proves the reliability of this method under the conditions established for the analysed case.

Chapter 6

Discussion

The research described in this draft and the whole development and design process contained in the previous chapters of this document are focused on answering the main questions stated in section 2. At this point, all the efforts made and represented in this project have a reliable prediction system for cyber attackers behaviour as a result, which can represent an answer to these questions.

In this case, the environment for which this method is designed is a **cyber range**. Learning from the information and being able to understand and measure how likely is an attacker to perform an action in a certain system can represent a very useful way to find vulnerabilities and protect it. When there's enough information available, a probabilistic graphical model like the one proposed here turns out to be an accurate method to trace and **understand** the behaviour of cyber attackers. The fact that Bayesian networks are not a simple graphical model but a reasoning method, represents a useful property for probabilistic prediction in this case. This type of probabilistic model, applied to a situation with special properties like the one described in this project, can be also applied to different datasets that also contain patterns based on events ordered in time.

This proposal represents a useful tool for fully controlled environments, but presents special weaknesses if there's a lack of information. In case of a cyber range where a set of cyber attacks can be detected and analysed, this model is able to perform accurate calculations, but this special situation is not common, and sometimes not even useful for the goal of a cyber range, which is meant to be an environment where challenges and problems can be solved in order to learn and improve.

The good performance of this model depends on a set of detailed infor-

mation about detected events. Cyber attacks detection represents an issue for which many projects are developed nowadays. Cyber attacks are usually hard to detect and trace, and this problem becomes exponentially more difficult when trying to analyse their details. It's usual that a cyber attacker performs an action in a system from another compromised host or uses any other technique to hide the source of the attack, which becomes a problem when trying to identify the responsible of the action. Other special aspects, like the challenge of understanding an action when some part of a system is found to be compromised, or analysing if an attack has been successful, failed or just a part of a larger process, have an strong presence in real cyber security challenges. Cyber attackers behaviour is usually very diverse, being sometimes impossible to classify actions as events, and all these factors make this project an unuseful method for a real case.

Due to the properties of a probabilistic graphical model like a **Bayesian network**, the performance and applicability of this model are higher in the case of a cyber range, for which it is specifically designed, as explained in chapter 4. This model represents the actions of cyber attackers as objects that influence the existence of the others, which provides a deeper understanding of their behaviour. Depending on the situation, this proposal can represent a very useful tool to find solutions and make decisions regarding to cyber security actions when problems are known and can be analysed, but not when the cyber range requires a deeper learning and there's not a reliable detection system, which is a very common case.

Although this model is designed for cyber attackers patterns, the versatility of Bayesian networks can make it a good solution for different approaches, representing this a useful proposal for an starting point to improve from.

Chapter 7

Conclusions

Cyber attackers behaviour tracing and prediction represents a problem of which solutions have an strongly innovative nature. In this case, the main purpose is associated not only with representation, but also with **understanding**. Probabilistic models are the base for most of the cyber security predictive methods, but there's still a clear lack of knowledge about this field that represents an obvious need for research.

Probabilistic graphical models can be an efficient tool to represent relations and properties of some set of elements given a discrete set of information. Applying probability concepts, specially Bayesian models, it's possible to accurately calculate and represent the nature of some system from discrete data using probabilities, like behavioural patterns. In this case, this type of graphical models are an effective way to trace and predict cyber attackers behaviour, due to their strong capability to represent causality and influence measures. Inside this scope, Bayesian networks represent a complex and reliable probabilistic model. In this case, applying this theory to cyber attackers actions, representing them as objects, it's possible to extract detailed and useful knowledge about them, but this model usually becomes inefficient when it's not applied in a correct environment, due to the large amount of information needed to get reliable results. This makes them become unuseful for a cyber range sometimes, and even more when being applied to a real case. Either way, Bayesian network theoretical models can have an strong usability for more specific cases and different approaches associated with cyber security prediction.

Based on basic probability concepts and Bayesian probability theory, a reliable model for pattern representation and analysis like the one designed and proposed here can be an efficient tool for a cyber range environment, to learn, understand and create **knowledge**. This contribution also represents a possi-

ble research starting point, having the possibility to create a more complete and efficient model based on this project, or being applied to other projects associated with a **cyber range**.

7.1 Future work

Although Bayesian networks have an strong capability to represent behavioural patterns, their application needs a large amount of information and some specific conditions that make them being discarded when being applied to a real case. The model proposed here is designed to be built from a dataset of detected attacks, also calculating predictions from them. Many aspects of this proposal can be improved in order to turn it into a more efficient and reliable system.

1. This model is designed to be applied for a dataset where the source and success of the actions can be detected. In order to be applied in a cyber range, an strong and reliable detection system is needed, for which a monitoring and filtering structure could be designed.

2. The possibility of using attack identifiers and success information is not usual for many cyber ranges, and it's also very far from a real case. Designing a learning method with a different structure learning algorithm, being able to find dependencies with the same reliability, would suppose a big improvement for this model.

3. The actual graph structure can be improved for a higher visual quality, representing the presence differences between variables or the strength of the connections.

4. The versatility of the model, based on a Bayesian network, also introduces the possibility to analyse more detailed and specific aspects of a cyber range, allowing to add more variables and causality relations that could help to understand and complete behavioural processes.

Bibliography

[1] Feng Xia et al. “Internet of things”. In: *International Journal of Communication Systems* 25.9 (2012), p. 1101.

[2] Mohamed Abomhara et al. “Cyber security and the internet of things: vulnerabilities, threats, intruders and attacks”. In: *Journal of Cyber Security and Mobility* 4.1 (2015), pp. 65–88.

[3] Dan Craigen, Nadia Diakun-Thibault, and Randy Purse. “Defining Cybersecurity”. In: *Technology Innovation Management Review* 4 (Oct. 2014), pp. 13–21. ISSN: 1927-0321. DOI: http://doi.org/10.22215/timreview/835. URL: http://timreview.ca/article/835.

[4] R. A. Kemmerer. “Cybersecurity”. In: *25th International Conference on Software Engineering, 2003. Proceedings.* May 2003, pp. 705–715. DOI: 10.1109/ICSE.2003.1201257.

[5] European Union Agency for Cybersecurity. “ENISA Threat Landscape Report 2017”. In: (2018).

[6] European Union Agency for Cybersecurity. “ENISA Threat Landscape Report 2018”. In: (2019).

[7] Andreea Bendovschi. “Cyber-attacks–trends, patterns and security countermeasures”. In: *Procedia Economics and Finance* 28 (2015), pp. 24–31.

[8] Barbara Kordy et al. “Foundations of Attack–Defense Trees”. In: *Formal Aspects of Security and Trust*. Ed. by Pierpaolo Degano, Sandro Etalle, and Joshua Guttman. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 80–95. ISBN: 978-3-642-19751-2.

[9] Arpan Roy, Dong Seong Kim, and Kishor S. Trivedi. "Attack countermeasure trees (ACT): towards unifying the constructs of attack and defense trees". In: *Security and Communication Networks* 5.8 (2012), pp. 929–943. DOI: 10.1002/sec.299. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sec.299. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.299.

[10] P. Larrañaga and S. Moral. "Probabilistic graphical models in artificial intelligence". In: *Applied Soft Computing* 11.2 (2011). The Impact of Soft Computing for the Progress of Artificial Intelligence, pp. 1511–1528. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2008.01.003. URL: http://www.sciencedirect.com/science/article/pii/S1568494608000094.

[11] A. G. Eleye-Datubo et al. "Enabling a Powerful Marine and Offshore Decision-Support Solution Through Bayesian Network Technique". In: *Risk Analysis* 26.3 (2006), pp. 695–721. DOI: 10.1111/j.1539-6924.2006.00775.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1539-6924.2006.00775.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1539-6924.2006.00775.x.

[12] Sarah Dorner, Jie Shi, and David Swayne. "Multi-objective modelling and decision support using a Bayesian network approximation to a non-point source pollution model". In: *Environmental Modelling Software* 22.2 (2007). Environmental Decision Support Systems, pp. 211–222. ISSN: 1364-8152. DOI: https://doi.org/10.1016/j.envsoft.2005.07.020. URL: http://www.sciencedirect.com/science/article/pii/S1364815205001817.

[13] Kristian Kristensen and Ilse A. Rasmussen. "The use of a Bayesian network in the design of a decision support system for growing malting barley without use of pesticides". In: *Computers and Electronics in Agriculture* 33.3 (2002), pp. 197–217. ISSN: 0168-1699. DOI: https://doi.org/10.1016/S0168-1699(02)00007-8. URL: http://www.sciencedirect.com/science/article/pii/S0168169902000078.

[14] P. Yermalovich and M. Mejri. "Formalization of Attack Prediction Problem". In: *2018 IEEE International Conference "Quality Management, Transport and Information Security, Information Technologies" (IT QM*

IS). Sept. 2018, pp. 280–286. DOI: 10.1109/ITMQIS.2018.8525128.

[15] Ekta Gandotra, Divya Bansal, and Sanjeev Sofat. "Computational Techniques for Predicting Cyber Threats". In: *Intelligent Computing, Communication and Devices*. Ed. by Lakhmi C. Jain, Srikanta Patnaik, and Nikhil Ichalkaranje. New Delhi: Springer India, 2015, pp. 247–253. ISBN: 978-81-322-2012-1.

[16] Vaclav Bartos et al. "Network entity characterization and attack prediction". In: *Future Generation Computer Systems* 97 (2019), pp. 674–686. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2019.03.016. URL: http://www.sciencedirect.com/science/article/pii/S0167739X18307799.

[17] Iffat A. Gheyas and Ali E. Abdallah. "Detection and prediction of insider threats to cyber security: a systematic literature review and meta-analysis". In: *Big Data Analytics* 1.1 (2016), p. 6. ISSN: 2058-6345. DOI: 10.1186/s41044-016-0006-0. URL: https://doi.org/10.1186/s41044-016-0006-0.

[18] Martin Husak et al. "Survey of Attack Projection, Prediction, and Forecasting in Cyber Security". In: *IEEE COMMUNICATIONS SURVEYS AND TUTORIALS* 21.1 (2019), 640–660. ISSN: 1553-877X. DOI: {10.1109/COMST.2018.2871866}.

[19] Yan Jia et al. "A Practical Approach to Constructing a Knowledge Graph for Cybersecurity". In: *Engineering* 4.1 (2018). Cybersecurity, pp. 53–60. ISSN: 2095-8099. DOI: https://doi.org/10.1016/j.eng.2018.01.004. URL: http://www.sciencedirect.com/science/article/pii/S2095809918301097.

[20] Jinsoo Shin et al. "Development of a cyber security risk model using Bayesian networks". In: *Reliability Engineering System Safety* 134 (2015), pp. 208–217. ISSN: 0951-8320. DOI: https://doi.org/10.1016/j.ress.2014.10.006. URL: http://www.sciencedirect.com/science/article/pii/S0951832014002464.

[21] X. Qin and W. Lee. "Attack plan recognition and prediction using causal networks". In: *20th Annual Computer Security Applications Conference*. Dec. 2004, pp. 370–379. DOI: 10.1109/CSAC.2004.7.

[22] Ahmet Okutan, Shanchieh Jay Yang, and Katie McConky. "Predicting Cyber Attacks with Bayesian Networks Using Unconventional Signals". In: *Proceedings of the 12th Annual Conference on Cyber and Information Security Research*. CISRC '17. Oak Ridge, Tennessee, USA: ACM, 2017, 13:1–13:4. ISBN: 978-1-4503-4855-3. DOI: 10.1145/3064814.3064823. URL: http://doi.acm.org/10.1145/3064814.3064823.

[23] J. Wu, L. Yin, and Y. Guo. "Cyber Attacks Prediction Model Based on Bayesian Network". In: *2012 IEEE 18th International Conference on Parallel and Distributed Systems*. Dec. 2012, pp. 730–731. DOI: 10.1109/ICPADS.2012.117.

[24] H. Holm et al. "P^{2}CySeMoL: Predictive, Probabilistic Cyber Security Modeling Language". In: *IEEE Transactions on Dependable and Secure Computing* 12.6 (Nov. 2015), pp. 626–639. ISSN: 1545-5971. DOI: 10.1109/TDSC.2014.2382574.

[25] O. Christou et al. "Phishing URL detection through top-level domain analysis: A descriptive approach". In: cited By 0. 2020, pp. 289–298. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85083039623&partnerID=40&md5=aea1ca3e05096ffbd2d90

[26] A.O. Almashhadani et al. "MaldomDetector: A system for detecting algorithmically generated domain names with machine learning". In: *Computers and Security* 93 (2020). cited By 0. DOI: 10.1016/j.cose.2020.101787. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85082944913&doi=10.1016%2fj.cose.2020.101787&partnerID=40&md5=dfe674adba28b277242cebfec6226c63.

[27] A. Sargolzaei et al. "Detection and Mitigation of False Data Injection Attacks in Networked Control Systems". In: *IEEE Transactions on Industrial Informatics* 16.6 (2020). cited By 0, pp. 4281–4292. DOI: 10.1109/TII.2019.2952067. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85081618470&doi=10.1109%2fTII.2019.2952067&partnerID=40&md5=5b7ba2972bdb6852a497a77cb80184b6.

[28] G. De La Torre Parra et al. "Detecting Internet of Things attacks using distributed deep learning". In: *Journal of Network and Computer Applications* 163 (2020). cited By 0. DOI: 10.1016/j.jnca.2020.102662. URL: https://www.scopus.com/inward/

record.uri?eid=2-s2.0-85083527952&doi=10.1016%2fj.jnca.2020.102662&partnerID=40&md5=b9d694b618964960abd59ba93b94d84a.

[29] M. Pawlicki, M. Choraś, and R. Kozik. "Defending network intrusion detection systems against adversarial evasion attacks". In: *Future Generation Computer Systems* 110 (2020). cited By 0, pp. 148–154. DOI: 10.1016/j.future.2020.04.013. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85083297694&doi=10.1016%2fj.future.2020.04.013&partnerID=40&md5=982254f131757af19e1ab7dc62a345fc.

[30] V.M. Bier. "Choosing what to protect". In: *Risk Analysis* 27.3 (2007). cited By 65, pp. 607–620. DOI: 10.1111/j.1539-6924.2007.00906.x. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-34447325155&doi=10.1111%2fj.1539-6924.2007.00906.x&partnerID=40&md5=aa914664fc39500ef684bb4f9

[31] R. Fang and X. Li. "A stochastic model of cyber attacks with imperfect detection". In: *Communications in Statistics - Theory and Methods* 49.9 (2020). cited By 0, pp. 2158–2175. DOI: 10.1080/03610926.2019.1568489. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85061058048&doi=10.1080%2f03610926.2019.1568489&partnerID=40&md5=ebe79f8d5c28e4a8165e2a2c768d8b0d.

[32] K. Hausken and G. Levitin. "Review of systems defense and attack models". In: *International Journal of Performability Engineering* 8.4 (2012). cited By 81, pp. 355–366. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84872381629&partnerID=40&md5=2398c71b855f5cbf8c86b23169e3ebd9.

[33] M.n. Azaiez. "A Bayesian Model for a game of information in optimal attack/defense strategies". In: *International Series in Operations Research and Management Science* 128 (2009). cited By 6, pp. 99–123. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84954501198&partnerID=40&md5=66c4b3708efd4be508b9d4b054215c50.

[34] Douglas Brent West et al. *Introduction to graph theory*. Vol. 2. Prentice hall Upper Saddle River, NJ, 1996.

[35] Béla Bollobás. *Modern graph theory*. Vol. 184. Springer Science & Business Media, 2013.

[36] Tyler J. VanderWeele and James M. Robins. “Directed Acyclic Graphs, Sufficient Causes, and the Properties of Conditioning on a Common Effect”. In: *American Journal of Epidemiology* 166.9 (Aug. 2007), pp. 1096–1104. ISSN: 0002-9262. DOI: 10.1093/aje/kwm179. URL: https://doi.org/10.1093/aje/kwm179.

[37] Jonathan L Gross and Jay Yellen. *Handbook of graph theory*. CRC press, 2004.

[38] H. A. Dawood. “Graph Theory and Cyber Security”. In: *2014 3rd International Conference on Advanced Computer Science Applications and Technologies*. Dec. 2014, pp. 90–96. DOI: 10.1109/ACSAT.2014.23.

[39] David B. Stephenson. “An Introduction to Probability Forecasting”. In: *Seasonal Climate: Forecasting and Managing Risk*. Ed. by Alberto Troccoli et al. Dordrecht: Springer Netherlands, 2008, pp. 235–257. ISBN: 978-1-4020-6992-5.

[40] Edwin T Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003.

[41] Robert B Ash. *Basic probability theory*. Courier Corporation, 2008.

[42] Kai Lai Chung. *A course in probability theory*. Academic press, 2001.

[43] Jay L Devore. *Probability and Statistics for Engineering and the Sciences*. Cengage learning, 2011.

[44] Peter Whittle. *Probability via expectation*. Springer Science & Business Media, 2012.

[45] José M Bernardo and Adrian FM Smith. *Bayesian theory*. Vol. 405. John Wiley & Sons, 2009.

[46] Richard T Cox. “Probability, frequency and reasonable expectation”. In: *American journal of physics* 14.1 (1946), pp. 1–13.

[47] Maurice J Dupré, Frank J Tipler, et al. “New axioms for rigorous Bayesian probability”. In: *Bayesian Analysis* 4.3 (2009), pp. 599–606.

[48] Irving John Good. “A derivation of the probabilistic explication of information”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 28.3 (1966), pp. 578–581.

[49] Ali Mohammad-Djafari. “Entropy, information theory, information geometry and Bayesian inference in data, signal and image processing and inverse problems”. In: *Entropy* 17.6 (2015), pp. 3989–4027.

[50] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[51] Todd Andrew Stephenson. "An Introduction to Bayesian Network Theory and Usage". In: (2000). URL: http://infoscience.epfl.ch/record/82584.

[52] Changhe Yuan, Heejin Lim, and Tsai-Ching Lu. "Most relevant explanation in Bayesian networks". In: *Journal of Artificial Intelligence Research* 42 (2011), pp. 309–352.

[53] Wim Wiegerinck, Willem Burgers, and Bert Kappen. "Bayesian networks, introduction and practical applications". In: *Handbook on Neural Information Processing*. Springer, 2013, pp. 401–431.

[54] Nir Friedman, Iftach Nachman, and Dana Peér. "Learning Bayesian Network Structure from Massive Datasets: The Candidate«Algorithm". In: *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*. UAI'99. Stockholm, Sweden: Morgan Kaufmann Publishers Inc., 1999, pp. 206–215. ISBN: 1-55860-614-9. URL: http://dl.acm.org/citation.cfm?id=2073796.2073820.

[55] Cassio Polpo de Campos and Qiang Ji. "Properties of Bayesian Dirichlet scores to learn Bayesian network structures". In: *Twenty-Fourth AAAI Conference on Artificial Intelligence*. 2010.

[56] Sergey Kirshner, Padhraic Smyth, and Andrew W Robertson. "Conditional Chow-Liu tree structures for modeling discrete-valued vector time series". In: *Proceedings of the 20th conference on Uncertainty in artificial intelligence*. AUAI Press. 2004, pp. 317–324.

[57] Harald Steck. "Constraint-Based Structural Learning in Bayesian Networks using Finite Data Sets". Dissertation. München: Technische Universität München, 2001.

[58] Changhe Yuan, Brandon Malone, and Xiaojian Wu. "Learning optimal Bayesian networks using A* search". In: *Twenty-Second International Joint Conference on Artificial Intelligence*. 2011.

[59] Ruoyong Yang and James O Berger. *A catalog of noninformative priors*. Institute of Statistics and Decision Sciences, Duke University, 1996.

[60] Harold Jeffreys. *The theory of probability*. OUP Oxford, 1998.

[61] *MITRE ATTCK Matrix for Enterprises*. https://attack.mitre.org/matrices/enterprise/.

[62] M. Ahmed and A.-S.K. Pathan. "False data injection attack (FDIA): an overview and new metrics for fair evaluation of its countermeasure". In: *Complex Adaptive Systems Modeling* 8.1 (2020). cited By 0. DOI: 10.1186/s40294-020-00070-w. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85083860018&doi=10.1186%2fs40294-020-00070-w&partnerID=40&md5=471d4d0481024701b3861aca0ec90218.

[63] F. Di Maio, R. Mascherona, and E. Zio. "Risk Analysis of Cyber-Physical Systems by GTST-MLD". In: *IEEE Systems Journal* 14.1 (2020). cited By 0, pp. 1333–1340. DOI: 10.1109/JSYST.2019.2928046. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85081690376&doi=10.1109%2fJSYST.2019.2928046&partnerID=40&md5=f79552e931e6276f953e21e1ab9f8eb

[64] Nicandro Cruz Ramırez. "Building Bayesian networks from data: a constraint based approach". PhD thesis. Citeseer, 2001.

[65] Timo JT Koski and John Noble. "A review of Bayesian networks and structure learning". In: *Mathematica Applicanda* 40.1 (2012).

[66] Harald Steck. "Constraint-Based Structural Learning in Bayesian Networks using Finite Data Sets". Dissertation. München: Technische Universität München, 2001.

[67] Tom Claassen and Tom Heskes. "A Bayesian approach to constraint based causal inference". In: *arXiv preprint arXiv:1210.4866* (2012).

[68] Diego Colombo and Marloes H Maathuis. "Order-independent constraint-based causal structure learning". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 3741–3782.

[69] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[70] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. *Graph theory with applications*. Vol. 290. Macmillan London, 1976.

[71] Richard E Neapolitan et al. *Learning bayesian networks*. Vol. 38. Pearson Prentice Hall Upper Saddle River, NJ, 2004.

[72] David Heckerman, Dan Geiger, and David M. Chickering. "Learning Bayesian networks: The combination of knowledge and statistical data". In: *Machine Learning* 20.3 (1995), pp. 197–243. ISSN: 1573-0565. DOI: 10.1007/BF00994016. URL: https://doi.org/10.1007/BF00994016.

[73] Gregory F. Cooper and Edward Herskovits. “A Bayesian method for the induction of probabilistic networks from data”. In: *Machine Learning* 9.4 (1992), pp. 309–347. ISSN: 1573-0565. DOI: 10.1007/BF00994110. URL: https://doi.org/10.1007/BF00994110.

[74] Sanjoy Dasgupta. “Learning polytrees”. In: *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 1999, pp. 134–141.

[75] Thomas Dyhre Nielsen and Finn Verner Jensen. *Bayesian networks and decision graphs*. Springer Science & Business Media, 2009.

[76] Alexandr A. Borovkov. *Probability Theory*. Springer, 2019.

[77] Herman Bruyninckx. “Bayesian probability”. In: *CiteSeer, não publicado em con* (2002), p. 81.

[78] Paul Damien et al. *Bayesian theory and applications*. OUP Oxford, 2013.

[79] Eugene Charniak. “Bayesian networks without tears.” In: *AI magazine* 12.4 (1991), pp. 50–50.

[80] Ana M MartıNez-RodrıGuez, Jerrold H May, and Luis G Vargas. “An optimization-based approach for the design of Bayesian networks”. In: *Mathematical and Computer Modelling* 48.7-8 (2008), pp. 1265–1278.

[81] Peter Spirtes and Christopher Meek. “Learning Bayesian networks with discrete variables from data.” In: *KDD*. Vol. 1. 1995, pp. 294–299.

Appendix A

Method code

```
# Import modules

import dash
import dash_table
import pandas as pd
import requests
import json
import math
import collections
import operator
from AnalysisModules import *
from datetime import datetime
from datetime import timedelta
from datetime import date
from random import choice
import dash_bootstrap_components as dbc
import dash_table_experiments as dte
import dash_core_components as dcc
import dash_html_components as html
import plotly.plotly as py
import plotly.graph_objs as go
import networkx as nx
import dash_cytoscape as cyto
from dash.dependencies import Input, Output, State
from flask import request

### Bayesian network construction

# Load extra layouts
cyto.load_extra_layouts()

```

```
#Creation of the classes and the principal list of moves
class attack:
  attacker = None # Identifier of the attacker
  name = None # Name of the action
  host = None # Attacked host
  datetime = None # Date and time when the action was performed
  success = None # Success of the attack

class state:
  name = None # Name of the node
  p_succ = None # Marginal probability of success
  p_fail = None # Marginal probability of failure
  parents = None # Parents of the node
  cpt = None # Conditional Probability Table
  pred = None # Table of calculated inferences

print("Moves list created")
#moves.sort(key=lambda x: x.datetime)
# First of all, the nodes list must be created
states = []
aux2 = []

# Every attack made to every host is a discrete distribution with two possible outcomes, success or failure
# These discrete distributions are treaten like an state, and every state has its own distribution and name
for x in moves:
  aux1 = []
  if not states:
    for y in moves:
      if y.name == x.name and y.host == x.host:
        aux1.append(y.success)
    new_state = state()
    new_state.p_succ = prob_success(aux1)
    new_state.p_fail = prob_fail(aux1)
    new_state.name = x.name + ' - ' + x.host
    aux2.append(new_state.name)
    states.append(new_state)
  name = x.name + ' - ' + x.host
  if name not in aux2:
    for y in moves:
      if y.name == x.name and y.host == x.host:
        aux1.append(y.success)
```

```
73      new_state = state()
74      new_state.p_succ = prob_success(aux1)
75      new_state.p_fail = prob_fail(aux1)
76      new_state.name = x.name + ' - ' + x.host
77      aux2.append(new_state.name)
78      states.append(new_state)
79
80  # Once the list of states is created, it's
      necessary to create a model of the network
81  print("States list created")
82
83  # It is necessary to set the dependencies between
      the nodes
84  # This is made comparing the typical behaviour of
      the attackers in order to find patterns
85  # Every pattern that is found is represented with
      an edge between parents and sons
86
87  attackers = []
88
89  # List of attackers ids
90  for move in moves:
91    if not attackers:
92      attackers.append(move.attacker)
93    else:
94      if move.attacker not in attackers:
95        attackers.append(move.attacker)
96
97  # Separate performed attacks by attacker id
98
99  samples = dict()
100
101 if len(attackers)>0:
102   for attacker in attackers:
103     samples[attacker] = []
104     for move in moves:
105       if move.attacker == attacker:
106         samples[attacker].append(move)
107     samples[attacker].sort(key=lambda x: x.datetime
      )
108
109 # Due to the success constraint (The next step can'
      t be reached after an unsuccessful attack),
      dependencies are calculated using successful
      attacks
110
111 samples_s = dict()
```

```
112
113 if len(attackers)>0:
114   for attacker in attackers:
115     samples_s[attacker] = []
116     for move in moves:
117       if move.attacker == attacker:
118         if move.success == 1:
119           samples_s[attacker].append(move)
120     samples_s[attacker].sort(key=lambda x: x.
    datetime)
121
122 # Make parent list for every state
123 for x in states:
124   x.parents = []
125   x.parents = findparents(x, samples_s, attackers)
126
127 filterroots(states,samples,attackers)
128 removecycles(states)
129
130 # Calculate Conditional Probability Tables for
    every state
131
132 for state in states:
133   if len(state.parents)>0:
134     state.cpt = cond_prob_table(state.name, state.
    parents, samples, attackers)
135   else:
136     state.cpt = dict()
137     state.cpt[state.name+'^']=state.p_fail
138     state.cpt[state.name]=state.p_succ
139
140 # Apply Bayes theorem and create inference values
    for diagnosis tables
141
142 for state in states:
143   state.pred = BT_diagnosis(state, states)
144
145 # The states must be configured as nodes and added
    to the network
146
147 app = dash.Dash(__name__, static_folder='C:\\Users
    \\pabca\\OneDrive\\Documentos\\Trabajo\\KTH\\
    Tesis\\App\\assets')
148
149 app.config['suppress_callback_exceptions']=True
150
151 nodes = []
```

```
for state in states:
  node = {'data': {'id': state.name,'name': state.
   name}}
  nodes.append(node)

# Create edges between parents and sons
edges = []

for x in states:
  if len(x.parents) >= 1:
    for y in x.parents:
      edge = {'data': {'source': y, 'target': str(x
   .name),'label': 'Node '+y+' to '+str(x.name)}}
      edges.append(edge)

elements = nodes + edges

# Define roots of the graph

roots = None

for x in states:
  if not x.parents:
    if roots == None:
      roots = '#'+x.name
    else:
      roots += ',#'+x.name

# Define style of the graph

default_stylesheet = [
    {
        "selector": 'node',
        'style': {
            "opacity": 1,
            'z-index': 9999,
            'font-family': 'helvetica',
            'font-size': 14,
            "content": "data(name)",
            "text-valign": "top",
            "text-halign": "center",
            "background-color": "#555",
            "text-outline-color": "#555",
            "text-outline-width": 2,
            "color": "#fff",
            "widht": 100,
            "overlay-padding": 6
```

```
            }
        },
        {
            "selector": 'edge',
            'style': {
                "curve-style": "bezier",
                "opacity": 0.45,
                'target-arrow-shape': 'triangle',
                'z-index': 5000
            }
        },
        {
            'selector': 'node:selected',
            "style": {
                "border-width": 10,
                "border-color": "crimson",
                "border-opacity": 0.3,
                "background-color": "#8A8A8A",
                "text-outline-color": "#464545"
            }
        },
        {
            "selector": 'node:hover',
            'style': {
                "background-color": "#8A8A8A"
            }
        },
        {
            "selector": "node.unhighlighted",
            "style": {
                "opacity": 0.2
            }
        },
        {
            "selector": "edge.unhighlighted",
            "style": {
                "opacity": 0.05
            }
        },
        {
            "selector": ".highlighted",
            "style": {
                "z-index": 999999
            }
        },
        {
            "selector": "node.highlighted",
```

```
        "style": {
            "border-width": 6,
            "border-color": "#AAD8FF",
            "border-opacity": 0.5,
            "background-color": "#394855",
            "text-outline-color": "#394855"
        }
    }
]

# Create graph layout
app.layout = html.Div([
    cyto.Cytoscape(
        id='cytoscape',
        elements=elements,
        style={'width': '100%', 'height': '350px'},
        stylesheet=default_stylesheet,
        layout={'name': 'klay', 'roots':roots}
    ),
    html.H4(id='cytoscape-tapNodeData-output_1'),
    html.Table(id='cytoscape-tapNodeData-output_2')
    ,
    html.H4(id='cytoscape-tapNodeData-output_3'),
    html.Table(id='cytoscape-tapNodeData-output_4')
    ,
    html.H4(id='cytoscape-tapEdgeData-output_1'),
    html.Table(id='cytoscape-tapEdgeData-output_2')
])

@app.callback(Output('cytoscape-tapNodeData-
    output_1','children'),
              [Input('cytoscape','tapNodeData')])
def displayTapNodeData_header_diagnosis(data):
  return 'Diagnosis table for "'+ data['id']+'"'

@app.callback(Output('cytoscape-tapNodeData-
    output_2','children'),
              [Input('cytoscape','tapNodeData')])
def displayTapNodeData_diagnosis(data):
  for state in states:
    if state.name == data['name']:
      if state.pred == None:
        return 'No information available.'
      else:
        return html.Table([html.Tr([html.Th('- P('+
    key+'):')]+[html.Td(round(state.pred[key],3))])
    for key in state.pred])
```

```
285
286 @app.callback(Output('cytoscape-tapNodeData-
    output_3','children'),
287               [Input('cytoscape','tapNodeData')])
288 def displayTapNodeData_header_cpt(data):
289   return 'Conditional probability table for "'+
    data['id']+'"'
290
291 @app.callback(Output('cytoscape-tapNodeData-
    output_4','children'),
292               [Input('cytoscape','tapNodeData')])
293 def displayTapNodeData_cpt(data):
294   for state in states:
295     if state.name == data['name']:
296       if state.cpt == None:
297         return 'No information available.'
298       else:
299         return html.Table([html.Tr([html.Th('- P('+
    key+'):')]+[html.Td(round(state.cpt[key],3))])
    for key in state.cpt])
300
301 @app.callback(Output('cytoscape-tapEdgeData-
    output_1','children'),
302               [Input('cytoscape','tapEdgeData')])
303 def displayTapEdgeData_header(data):
304   return 'Found dependency between "'+ data['source
    ']+'" and "'+ data['target']+'"'
305
306 @app.callback(Output('cytoscape-tapEdgeData-
    output_2','children'),
307               [Input('cytoscape','tapEdgeData')])
308 def displayTapEdgeData_prob(data):
309   for state in states:
310     if state.name == data['target']:
311       array = state.cpt
312       key = max(array, key=lambda x: array.get(x))
313       return html.Table([html.Tr([html.Th('- P('+
    key+'):')]+[html.Td(str(round(array[key],3)))])
    ])
314
315 if __name__ == '__main__':
316     app.run_server(debug=False)
```

Appendix B

Functions and mathematical operations code

```
# AnalysisModules.py
# The processes applied to detect attacks and relevant actions from attackers are defined in this script

# Import modules
import math
import random
import itertools
from datetime import datetime
from datetime import timedelta
from datetime import date

# Given a list of results, calculates the marginal probability of success
def prob_success(samplelist):
  success = 0
  for x in samplelist:
    if x == 1:
      success += 1
  prob = success/len(samplelist)
  return prob

# Given a list of results, calculates the marginal probability of failure
def prob_fail(samplelist):
  fail = 0
  for x in samplelist:
    if x == 0:
```

```
      fail += 1
  prob = fail/len(samplelist)
  return prob

# Given an state, a list of samples and a list of attackers IDs, find dependencies by comparing patterns
def findparents(state, sampleslist, attackerslist):
  class pos:
    name = None
    presence = 0
  possibilities = []
  parents = []
  possibilities_n = []
  max_p = 0
  for attacker in attackerslist:
    if len(sampleslist[attacker])>0:
      for x in range(len(sampleslist[attacker])):
        name = sampleslist[attacker][x].name+' - '+sampleslist[attacker][x].host
        if name == state.name:
          if x>0:
            if not possibilities_n:
              new_pos = pos()
              new_pos.name = sampleslist[attacker][x-1].name+' - '+sampleslist[attacker][x-1].host
              new_pos.presence = 1
              max_p = 1
              possibilities.append(new_pos)
              possibilities_n.append(new_pos.name)
            if sampleslist[attacker][x-1].name+' - '+sampleslist[attacker][x-1].host not in possibilities_n:
              new_pos = pos()
              new_pos.name = sampleslist[attacker][x-1].name+' - '+sampleslist[attacker][x-1].host
              new_pos.presence = 1
              max_p += 1
              possibilities.append(new_pos)
              possibilities_n.append(new_pos.name)
            if sampleslist[attacker][x-1].name+' - '+sampleslist[attacker][x-1].host in possibilities_n:
              for possibility in possibilities:
                if possibility.name == sampleslist[attacker][x-1].name+' - '+sampleslist[attacker][x-1].host:
```

```
62                    possibility.presence += 1
63                    max_p += 1
64    if len(possibilities)>0:
65      for possibility in possibilities:
66        if possibility.presence < 3:
67          possibilities.remove(possibility)
68      for possibility in possibilities:
69        parents.append(possibility.name)
70
71    return parents
72
73  # Filter the possible root nodes of the graph
74  # Checks the first attack made by every attacker
75  def filterroots(stateslist, sampleslist,
      attackerslist):
76    for state in stateslist:
77      for attacker in attackerslist:
78        if len(sampleslist[attacker])>0:
79          if state.name == sampleslist[attacker][0].
      name+' - '+sampleslist[attacker][0].host:
80            state.parents = []
81
82  # Removes cycles and redundant dependencies (
      Dependencies can only have one direction)
83  def removecycles(stateslist):
84    for state in stateslist:
85      if state.name in state.parents:
86        state.parents.remove(state.name)
87      if len(state.parents)>0:
88        for parent in state.parents:
89          for s in stateslist:
90            if s.name == parent:
91              if state.name in s.parents:
92                s.parents.remove(state.name)
93
94  # Creates the conditional probability table of a
      node given its parents, a list of samples and a
      list with attackers IDs
95  def cond_prob_table(son, parents, sampleslist,
      attackers):
96    cpt_s = dict()
97    comb = list(itertools.product([0, 1], repeat=len(
      parents)))
98
99    for combination in comb:
100     aux = []
101     parents_s = None
```

```
102     for attacker in attackers:
103       if len(sampleslist[attacker])>0:
104         for x in range(len(sampleslist[attacker])):
105           if sampleslist[attacker][x].name+' - '+
    sampleslist[attacker][x].host == son:
106             if x>0:
107               if sampleslist[attacker][x-1].name+'
    - '+sampleslist[attacker][x-1].host in parents:
108                 if sampleslist[attacker][x-1].
    success == combination[parents.index(sampleslist
    [attacker][x-1].name+' - '+sampleslist[attacker
    ][x-1].host)]:
109                   aux.append(sampleslist[attacker][
    x].success)
110     for x in range(len(parents)):
111       if parents_s == None:
112         if combination[x] == 0:
113           parents_s = parents[x]+'^'
114         elif combination[x] == 1:
115           parents_s = parents[x]
116       else:
117         if combination[x] == 0:
118           parents_s += ','+parents[x]+'^'
119         elif combination[x] == 1:
120           parents_s += ','+parents[x]
121     if len(aux)>0:
122       for x in range(2):
123         if x == 0:
124           cpt_s[son+'^|'+parents_s] = aux.count(x)/
    len(aux)
125         if x == 1:
126           cpt_s[son+'|'+parents_s] = aux.count(x)/
    len(aux)
127     else:
128       for x in range(2):
129         if x == 0:
130           cpt_s[son+'^|'+parents_s] = 0
131         if x == 1:
132           cpt_s[son+'|'+parents_s] = 0
133   return cpt_s
134
135 # Computes the normalization constant of an state
136 # The normalization constant of an state is the
    probability of that state to be TRUE or FALSE (
    Depending on the situation) in all cases
137 def norm_cons(name, stateslist):
138   cons = 0
```

```
139
140 for state in stateslist:
141   if state.name in name:
142     if len(state.parents) == 1:
143       for parent in state.parents:
144         for state_s in stateslist:
145           if state_s.name == parent:
146             cons += state.cpt[name+'|'+parent]*
    state_s.p_succ
147             cons += state.cpt[name+'|'+parent+'^'
    ]*state_s.p_fail
148     if len(state.parents) > 1:
149       comb = list(itertools.product([0, 1],
    repeat=len(state.parents)))
150       for combination in comb:
151         s_names = []
152         for parent in state.parents:
153           if combination[state.parents.index(
    parent)] == 0:
154             s_names.append(parent+'^')
155           if combination[state.parents.index(
    parent)] == 1:
156             s_names.append(parent)
157         for key in state.cpt:
158           aux = True
159           eq = 1
160           if name in key:
161             if '^' not in name:
162               if (name+'^') not in key:
163                 for s_name in s_names:
164                   if s_name not in key:
165                     aux = False
166                   if '^' not in s_name:
167                     if (s_name+'^') in key:
168                       aux = False
169                 if aux == True:
170                   eq *= state.cpt[key]
171                   print (key)
172                   for s_name in s_names:
173                     for state_s in stateslist:
174                       if state_s.name in s_name:
175                         if '^' in s_name:
176                           eq *= state_s.p_fail
177                           print(state_s.name+'^')
178                         else:
179                           eq *= state_s.p_succ
180                           print(state_s.name)
```

```
                    else:
                      eq = 0
                    cons += eq
                else:
                  for s_name in s_names:
                    if s_name not in key:
                      aux = False
                    if '^' not in s_name:
                      if (s_name+'^') in key:
                        aux = False
                  if aux == True:
                    eq *= state.cpt[key]
                    print (key)
                    for s_name in s_names:
                      for state_s in stateslist:
                        if state_s.name in s_name:
                          if '^' in s_name:
                            eq *= state_s.p_fail
                            print(state_s.name+'^')
                          else:
                            eq *= state_s.p_succ
                            print(state_s.name)
                  else:
                    eq = 0
                  cons += eq
  if cons > 1:
    cons = 1
  if cons < 0:
    cons = 0
  return cons

# Computes the inference and find predictive dependencies of the steps given the observed events
# The treated events are conditionally independent
# When there's more than one son, the joint conditional probability is a multiplications of the conditional probabilities of every son
def BT_diagnosis(node, stateslist):
  pred_s = dict()
  sons = []

  for state in stateslist:
    if node.name in state.parents:
      sons.append(state.name)
  if not sons:
    pred_s = None
```

```
224   else:
225     comb = list(itertools.product([0, 1], repeat=
      len(sons)))
226     for x in range(2):
227       if x == 0:
228         n_name = node.name+'^'
229         for combination in comb:
230           joint_p = 1
231           constant = 1
232           s_names = []
233           p_name = n_name+'|'
234           for y in range(len(combination)):
235             if combination[y] == 0:
236               s_names.append(sons[y]+'^')
237             if combination[y] == 1:
238               s_names.append(sons[y])
239           for son in sons:
240             for state in stateslist:
241               if state.name == son:
242                 if len(state.parents)==1:
243                   print("Joint probability of "+
      s_names[sons.index(son)]+'|'+n_name)
244                   joint_p = state.cpt[s_names[sons.
      index(son)]+'|'+n_name]
245                   print(joint_p)
246                 else:
247                   print('Joint probability of '+
      n_name+' and '+s_names[sons.index(son)])
248                   for key in state.cpt:
249                     if n_name in key:
250                       if s_names[sons.index(son)]
      == son:
251                         if s_names[sons.index(son)]
       in key and s_names[sons.index(son)]+'^' not in
      key:
252                           aux = s_names[sons.index(
      son)]+n_name
253                           counter = aux.count('^')
254                           if key.count('^') ==
      counter+(len(state.parents)-1):
255                             print(key)
256                             print(state.cpt[key])
257                             joint_p *= state.cpt[
      key]
258                             print(joint_p)
259                             for parent in state.
      parents:
```

```
260                                 if parent not in
    n_name:
261                                   for state_s in
    stateslist:
262                                     if state_s.name
    == parent:
263                                       joint_p *=
    state_s.p_fail
264                         else:
265                           if s_names[sons.index(son)]
     in key:
266                             aux = s_names[sons.index(
    son)]+n_name
267                             counter = aux.count('^')
268                             if key.count('^') ==
    counter+(len(state.parents)-1):
269                               print(key)
270                               print(state.cpt[key])
271                               joint_p *= state.cpt[
    key]
272                               print(joint_p)
273                               for parent in state.
    parents:
274                                 if parent not in
    n_name:
275                                   for state_s in
    stateslist:
276                                     if state_s.name
    == parent:
277                                       joint_p *=
    state_s.p_fail
278           for s in s_names:
279             if s_names.index(s) == 0:
280               p_name += s
281             if s_names.index(s) > 0:
282               p_name += ','
283               p_name += s
284           for s in s_names:
285             print('Normalization constant of '+s)
286             constant *= norm_cons(s, stateslist)
287             print('Normalization constant of '+s+'
    is '+str(constant))
288           for state in stateslist:
289             if node.name in state.name:
290               prior = state.p_fail
291           print('The prior value is '+str(prior))
292           if constant == 0:
```

```
293             if joint_p == 0:
294               pred_s[p_name] = 0
295             else:
296               pred_s[p_name] = 1
297           else:
298             pred_s[p_name] = (joint_p*prior)/
    constant
299           print('Value of '+p_name+'is '+str(pred_s
    [p_name]))
300           if pred_s[p_name] > 1:
301             pred_s[p_name] = 1
302           if pred_s[p_name] < 0:
303             pred_s[p_name] = 0
304       if x == 1:
305         n_name = node.name
306         for combination in comb:
307           joint_p = 1
308           constant = 1
309           s_names = []
310           p_name = n_name+'|'
311           for y in range(len(combination)):
312             if combination[y] == 0:
313               s_names.append(sons[y]+'^')
314             if combination[y] == 1:
315               s_names.append(sons[y])
316           for son in sons:
317             for state in stateslist:
318               if son == state.name:
319                 if len(state.parents)==1:
320                   print("Joint probability of "+
    s_names[sons.index(son)]+'|'+n_name)
321                   joint_p *= state.cpt[s_names[sons
    .index(son)]+'|'+n_name]
322                   print(joint_p)
323                 else:
324                   print('Joint probability of '+
    n_name+' and '+s_names[sons.index(son)])
325                   for key in state.cpt:
326                     if n_name in key and n_name+'^'
     not in key:
327                       if s_names[sons.index(son)]
    == son:
328                         if s_names[sons.index(son)]
     in key and s_names[sons.index(son)]+'^' not in
    key:
329                           aux = s_names[sons.index(
    son)]+n_name
```

```
330                             counter = aux.count('^')
331                             if key.count('^') ==
    counter+(len(state.parents)-1):
332                               print(key)
333                               print(state.cpt[key])
334                               joint_p *= state.cpt[
    key]
335                               print(joint_p)
336                               for parent in state.
    parents:
337                                 if parent not in
    n_name:
338                                   for state_s in
    stateslist:
339                                     if state_s.name
    == parent:
340                                       joint_p *=
    state_s.p_fail
341                         else:
342                           if s_names[sons.index(son)]
     in key:
343                             aux = s_names[sons.index(
    son)]+n_name
344                             counter = aux.count('^')
345                             if key.count('^') ==
    counter+(len(state.parents)-1):
346                               print(key)
347                               print(state.cpt[key])
348                               joint_p *= state.cpt[
    key]
349                               print(joint_p)
350                               for parent in state.
    parents:
351                                 if parent not in
    n_name:
352                                   for state_s in
    stateslist:
353                                     if state_s.name
    == parent:
354                                       joint_p *=
    state_s.p_fail
355             if s_names.index(s) == 0:
356               p_name += s
357             if s_names.index(s) > 0:
358               p_name += ','
359               p_name += s
360           for s in s_names:
```

```
361             constant *= norm_cons(s, stateslist)
362             print('Normalization constant of '+s+'
    is '+str(constant))
363           for state in stateslist:
364             if node.name in state.name:
365               prior = state.p_succ
366           print('The prior value is '+str(prior))
367           if constant == 0:
368             if joint_p == 0:
369               pred_s[p_name] = 0
370             else:
371               pred_s[p_name] = 1
372           else:
373             pred_s[p_name] = (joint_p*prior)/
    constant
374           print('Value of '+p_name+'is '+str(pred_s
    [p_name]))
375           if pred_s[p_name] > 1:
376             pred_s[p_name] = 1
377           if pred_s[p_name] < 0:
378             pred_s[p_name] = 0
379   return pred_s
```

Appendix C

Styles

```
@import url(https://fonts.googleapis.com/css?family
   =Lato:100,300,400|Playfair+Display:400,700,400
   italic|Libre+Baskerville:400,700,400italic|Muli
   :300,400|Open+Sans:400,300,700|Oswald:400,700|
   Raleway:400,100,300,700|Montserrat:400,700|
   Merriweather:400,300,300italic,400italic,700|
   Bree+Serif|Vollkorn:400italic,400,700|Abril+
   Fatface|Cardo:400,400italic);

h4 {
  font-family: 'Lato';
  font-size: 1.5rem;
  font-weight: 900;
  text-align: center;
  text-shadow: 1px 1px #808080;
  color: #000;
}

h5 {
  font-family: 'Lato';
  font-size: 0.75rem;
  font-weight: 900;
  text-align: center;
}

p {
  font-family: 'Lato';
  font-size: 0.75rem;
  text-align: center;
}

```

```
table {
  margin-left: auto;
  margin-right: auto;
  border-collapse: collapse;
}

td {
  border-bottom: 0.5px solid black;
  font-family: 'Lato';
  text-align: center;
  padding: 10px
}

th {
  border-bottom: 2px solid black;
  border-right: 2px solid black;
  font-family: 'Lato';
  font-weight: bold;
  text-align: left;
  padding: 5px
}
```