POLITECNICO DI TORINO

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING (DAUIN)

Master Degree in Computer Engineering

Master Degree Thesis

Combining Ontologies and MulVAL to Generate Attack Bayesian Networks

Author: Stefano Iacono

Advisor: Alessandro Savino
Co-Advisor: Nicoló Maunero

April, 2025

Acknowledgments

A me stesso, per la forza e la voglia di non mollare le cose a cui tengo, nonostante le difficoltà.

A mio padre e a Rosetta, per essermi stati vicini con la mente e col cuore.

A Fabio, perché anche se diametricalmente opposti rimane un frammento del mio essere.

Ai miei zii, per avermi accompagnato fin qui senza mai un ripensamento e per la bellezza di dare loro questo piacere.

A Serena, che mi ha guardato dentro l'anima fino a toccare le corde che emettono le frequenze risonanti dei nostri esseri.

Ai miei amici, quelli sempre presenti che hanno capito quanto tengo a loro, anche quando ho attraversato momenti bui e avrei dovuto dare di più.

Al tempo, astratto e invisibile, allo spazio che colma i vuoti anche dove vi sono, alle coincidenze che ti fanno capire quanto è bella la vita e alla vita stessa.

Vi sono riconoscente, vi voglio bene.
Credi sempre.

Abstract

In today's interconnected digital world, assessing and managing cybersecurity risks, especially in the landscape of enterprises, becomes fundamental to equally protect human rights and business. Enterprises are by definition the best playhouse for cyber attackers, since information leakages or asset compromise could lead to catastrophic impacts over people and companies. Considering the enormous technological improvement of capabilities and assets, and their spreading all over the sectors, makes this task even more challenging.

Modern IT infrastructures continue growing in complexity, while cyber threats are constantly evolving to exploit vulnerabilities and find ways to spread across the interconnected assets within a system. This constant change makes it hard, even for experts, to study the companies' attack surfaces in order to predict, counter, and mitigate cyber threats from internal and external malicious actors. Security analysts are required with more and more efforts to correctly assess cybersecurity risk and to apply suitable countermeasures. Moreover, nowadays the huge amount of information to handle makes it almost impossible, or at least insufficient, to produce consistent predictions with modern tools, that usually make use of Intrusion Detection System's (IDS) logs to perform their jobs.

This thesis will present threat propagation and, more in detail, the semi-automatic detection of possible attack paths that can be followed by an attacker to reach a target. Many ideas and works exist in the literature regarding threat propagation, but all of these works focus only on specific steps of the process and often do not consider the synergy between them, which is crucial to improve both the efficiency and the effectiveness of the entire risk assessment process.

With this work, I want to open the door to possible implementation of a pipeline that consider three main steps: using ontology-based methods to model the IT infrastructure, with particular reference to the ThreMA project; using the open source tool MulVAL (Multihost, multistage Vulnerability Analysis), which is able to create attack graphs starting from properly converted infrastructure files and the provided rules for modeling attacks; and finally taking advantage of the Bayesian Network (BN) properties to build a structure, called Attack BN, that, given the appropriate probability inputs, can infer the propagation of an attack from a starting point to the target through the various intermediate nodes. This solution does not eliminate the need for human support, as the presence of experts feeding the tool is considered fundamental. The work of this thesis focused on the realization of an initial Proof-of-Concept of the Attack BN, focusing on part of the proposed pipeline and the generation of attack graphs using MulVAL. This

will provide a starting point for future studies and improvements leveraging already existing AI engines or reasoners that, along with other technologies, could fully automate the proposed pipeline.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, the growing sophistication of cyber threats has underscored the importance of Cybersecurity Risk Assessment as a foundational practice for safeguarding digital infrastructures. Modern organizations face a constantly landscape of threats, where attackers exploit vulnerabilities to disrupt systems, compromise data, and cause financial and reputational damage. With these threats becoming more diverse and persistent, understanding and mitigating the complex pathways through which an initial vulnerability can propagate across an entire network has become essential.

As technology and methodologies rapidly advance, the associated threats and vulnerabilities also evolve. In this fast-paced environment, people often are not able to prevent malicious events proactively, which can lead to catastrophic outcomes. As this sentence can be applied for several fields where the mankind is involved, the cybersecurity one is not outdone.

The wide spreading of cyber threats and the always more frequent attacks to IT infrastructures lead the researchers to invest their time to find a solution to avoid, or at least significantly reduce, the impact they may have over systems and people, in order to protect both economical interests but also the human rights, against the so called *zero-days*, namely those vulnerabilities that are not well known to vendors and from which it is possible to defend applying some predictive mechanisms, since no patches to fix the issue have been released yet. Such predictions are usually based on a paradigm called *Technique, Tactics and Procedure* (TTP) [16], aimed to analyse the potential behaviour of an attacker or, more in general, of a threat actor, by including into the analysis the several steps involved into an attack and other observations that can give important information about possible flaws of the IT systems.

Cybersecurity risk assessment methodologies vary significantly, from asset-centric approaches focused on evaluating critical systems, to threat-focused assessments that anticipate attacker behaviour, to vulnerability-based scans that detect specific weaknesses. Despite the advancements offered by each approach, existing methodologies face key limitations in scalability, adaptability, and accuracy. These issues become particularly evident in complex systems where the interconnectedness of assets and threats creates highly dynamic attack surfaces. Traditional methods often struggle to account for these relationships comprehensively, limiting their effectiveness in predicting how threats may spread

from one component to another. Moreover, many current solutions are dependent on the expertise of analysts, which, while valuable, introduces potential for inconsistency and overlooks hidden threats in large-scale or rapidly changing systems.

An emerging response to these challenges is the ontology-based approach, which leverages structured knowledge representation to establish formal relationships between assets, threats, vulnerabilities, and security controls. By creating a semantic model that connects various components within a system, ontology-based cybersecurity risk assessment enables a more systematic and automated threat propagation analysis. This structure can enhance the detection of hidden attack pathways, making it possible to infer new relationships that may not be immediately apparent. However, while promising, even ontology-based models often lack robust mechanisms for predicting threat evolution dynamically or quantifying the probability of different attack paths in real time.

Building upon the foundation of ontology-based assessment, this thesis proposes a novel approach to automating threat propagation analysis by integrating Bayesian Network modelling. Our goal is to enhance the accuracy and scalability of threat prediction by creating a probabilistic model that can dynamically evaluate the likelihood of various attack paths. This model goes beyond static representation, allowing for real-time updates as new threats, vulnerabilities, or interconnections are discovered within the system. By leveraging Bayesian Networks, we can quantify the conditional probabilities associated with each threat path, providing a more nuanced assessment of potential risks.

The contribution of this research is twofold: first, it aims to address the limitations of existing cybersecurity risk assessment approaches by automating threat propagation predictions, reducing reliance on manual analysis; second, it introduces a probabilistic framework that can adapt to new threats and attack techniques, ultimately providing organizations with a more resilient and responsive risk assessment tool.

The remainder of this document is organized as follows: Chapter 2 provides the necessary background, outlining the core concepts of cybersecurity risk assessment methodologies and their respective strengths and limitations. Chapter 3 presents related works, analysing current solutions and identifying areas for improvement. Chapter 4 details the proposed Bayesian Network model for threat propagation, explaining its structure, advantages, and potential applications. Chapters 5 and 6 discuss experimental results, validate our approach, and conclude with future directions.

Chapter 2

Background

The purpose of this chapter is to give the basic understanding for the cybersecurity risk assessment and to explore the landscape of the latter, including the role of attack graphs and in understanding the threat propagation.

2.1 Cybersecurity Risk Assessment Approaches

A cybersecurity risk assessment is a systematic process with the scope to identify vulnerabilities and threats within an organization's IT environment, evaluating the likelihood of a security event and the potential impact of the latter on the *CIA triad* (Confidentiality, Integrity, Availability) [4], so to identify the risks to organizational assets, individuals, other organizations, and the Nation, resulting from the operation of an information system. The CIA triad is used as model designed to guide experts in securing the systems involved in an IT infrastructure and it represents the assets' properties to be protected, depending on the typology of the asset itself, and providing the possible targets of threats or attackers exploiting vulnerabilities. This process typically provides guidance and recommendations for implementing additional security controls designed to counter the organization's threats. In particular, these controls aim to reduce the risk stemming from the exploitation of vulnerabilities, that is, any weakness in an information system, its security procedures, internal controls, or implementation that could be exploited by a threat to trigger a security event.

The paper written by Paulsen [25] gives a complete list of the terminology definitions typically used in the cybersecurity context, including the description of assessment, CIA triad, vulnerability and everything else can be applied to the cybersecurity landscape.

The cybersecurity risk assessment process is composed of multiple interconnected steps, that are generally followed to perform it:

1. **Scope Definition**: establish the boundaries and objectives of the assessment by identifying the systems, processes, and assets to be evaluated.

2. **Asset Identification**: catalog critical assets, such as hardware, software, data, and human resources, that support the organization's operations.

3. **Threat Identification**: recognize potential threats, including cyberattacks, natural disasters, and insider risks, which could compromise the identified assets.

4. **Vulnerability Identification**: detect weaknesses in systems, procedures, and controls that may be exploited by the identified threats.

5. **Risk Analysis**: assess the likelihood of threat exploitation and the potential impact on the confidentiality, integrity, and availability (CIA) of assets.

6. **Risk Evaluation**: prioritize risks based on their severity and the potential business impact, guiding where immediate attention is required.

7. **Risk Mitigation**: develop and implement strategies, such as enhanced security controls or disaster recovery plans, to reduce or eliminate the identified risks.

8. **Monitoring and Review**: continuously oversee the risk environment and the effectiveness of mitigation measures, updating the assessment as needed.

However, there are several approaches for experts to evaluate risks, and the decision of which approach is the best is usually controversial. In fact, each approach differs on the order of those steps and on which step they focus more, in order to emphasize different aspects. Of course, each one has pros and cons, and thus experts often choose the approach based on their experience and knowledge.

Contextually, cybersecurity risk assessment approaches are many and differ on what they are focusing on as basement for the risk analysis, even if they pass through the same elements of the process, but the most used are the following [27]:

- the **asset/impact-oriented** approach, in which the purpose of the assessment is to firstly identify which are the primary and secondary assets of a company (e.g. server, data, networks and so on), such that they are important for the company's stakeholder, meaning they may compromise some economical and legal aspects. Assets are prioritized following the economic impact they have for the company and at the same time which of them disrupt the business continuity in case they are compromised. With this approach, for each of the identified asset, the focus is on studying the factors exposing those critical assets to possible compromising to reach some goals, along with the evaluation of the possible impact of those attacks on the CIA triad on several fields of interest for the company. Vulnerabilities are usually identified by using the support of online platform as the NVD[1]. At the end, an expert is able to define which are the controls to be applied to each asset, in order to mitigate the threats or, at least, to reduce the risk of the threat to compromise the asset. This approach is generally used because of the focus on the assets, so that the most critical can be paid attention, but they may not keep trace of possible external threats, since the attention on the internal assets may not consider zero-days or new attackers' techniques and tactics.

[1] https://nvd.nist.gov/

- the **threat-oriented** approach, in which the focus is moved to the identification of both internal and external threats that may compromise the system, rather than the assets themselves. With this approach, it is easier for the expert to try to predict attackers' Techniques, Tactics and Procedures (TTP), since known and potential attacks are studied and identified and placed as starting point for the vulnerabilities individuation and the evaluation of their impacts. It usually relies on frameworks such as CAPEC[2] and MITRE ATT&CK[3], how threats can develop throughout the IT environment. In particular, CAPEC provides a catalogue of *common attack patterns*, showing how adversaries are able to exploit weaknesses of applications and giving to the users a way to analyse threats using a standardized way, enumerating and describing them by providing the prerequisites, the consequences, the likelihood and the skill required by the attacker, along with the description of the pattern itself. Usually, it is used in combination with MITRE ATT&CK, which provides a knowledge base about the attackers' TTP, both external and internal, giving to the experts information that can be useful to determine which are the most appropriate security controls to be applied and, furthermore, to proactively apply countermeasures to avoid attacks or to circumscribe them, rather than to be reactive, and so to apply countermeasures after the security event. This leads to the possibility to detect external threats and to mitigate even emerging ones, such as the zero-days, but at the same time it may open the door to some attacks targeting less critical assets or exploiting old vulnerabilities, or, even more, it can lead to the possibility to find false positives.

- the **vulnerability-oriented** approach, in which the first focus is moved to the specific vulnerability identification, in order to assess the risk by detecting the weaknesses of the system that can lead to the realization of a threat. Usually it relies on vulnerabilities scanners such as Nessus[4] and OpenVAS[5], that are able to analyse the system in a way to find well known vulnerabilities and often indexed taking advantage by the CVSS (Common Vulnerability Scoring System) [29]. It is usually used in combination with other approaches, since it is not able to identify emergent vulnerabilities or to detect new threats that use several vulnerabilities in combination, deviating the attention from the assets, and so leaving the possibility to underestimate the criticality of the latter.

All advantages and disadvantages of the above methods are summarized in Table 2.1, to give the reader a way to better understand the bullet points of each assessment method previously presented.

However, the list is not exhaustive, since many other approaches exist, even if they are less used or known, but it is enough to have the needed overview of the work behind the

[2] https://capec.mitre.org/

[3] https://attack.mitre.org/

[4] https://www.tenable.com/products/nessus

[5] https://www.openvas.org/

cybersecurity risk assessment and, at the same time, to avoid the reader to be overwhelmed by a very large amount of information, which would not be further useful in this document.

Assessment Approach	**Advantages**	**Disadvantages**
Asset-Oriented	• Focuses on critical assets. • Helps prioritize resources based on asset importance.	• Limited consideration of dynamic threat landscape. • May overlook vulnerabilities not directly linked to key assets.
Threat-Oriented	• Targets known threats, enabling proactive defences. • Adaptable to changing threat intelligence.	• Relies heavily on up-to-date threat intelligence. • May miss vulnerabilities unrelated to active threats.
Vulnerability-Oriented	• Identifies weaknesses that could be exploited. • Provides a detailed view of security gaps.	• Can result in overload of low-priority issues. • Less focus on specific threat scenarios or asset impact.

Table 2.1. Advantages and Disadvantages of Different Cybersecurity Assessment Approaches

2.2 Ontology and Its Applications

The ontological approach in the sciences has been widely used since the XVII century to bridge the gap between researchers' interpretations of reality, approving it. It facilitates the definition of a common language for discussing a specific topic, including the terminology and the relationships between the entities involved in an ontology. *Ontology* is a

philosophical term that can be applied to various scenarios and is particularly useful in science when complex subjects need to be modeled and discussed. It allows for the formalization of semantics and the rules governing the study environment, while also providing a cutting-edge representation of the underlying domain as the ontology evolves alongside the relevant literature. In fact, to develop an ontology, one must define the entities and the relationships between them, often based on rules and constraints that can also be specified by the user. Nowadays, numerous tools and descriptive languages are available to define ontologies from scratch, or even to build upon other related (or unrelated) ontologies. For example, one of the most famous ontology, developed by W3C[6] is the Wine Ontology, which employs one of the most used knowledge representation language for authoring ontologies, that will be briefly presented later in Chapter 3. Some tools are also able to make the ontology more human-readable, giving the possibility to represent it as a graph, hence made of nodes, the entities, and links, the relations, and showing how the entities of the ontology are involved into the connections, providing information about the relations acting as links between two nodes.

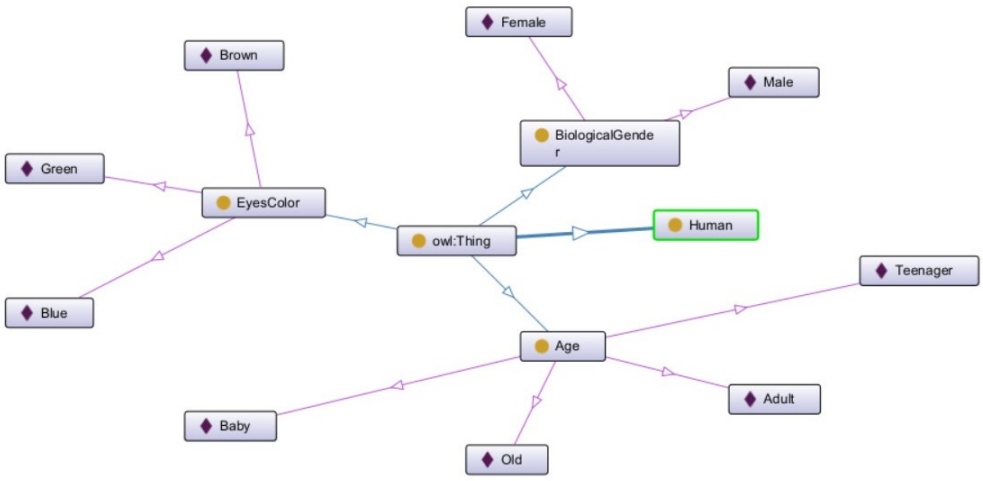

Figure 2.1. Example of visual representation of the Human Ontology.

Figure 2.1 shows a very simplified representation of the human ontology. From the point of view of the ontology, we can figure out three main characteristics in the graph, that in general are very close to Object Oriented (OO) languages and typical high level representation of classes:

[6] https://www.w3.org/

- the **Entities**, labeled with a yellow circle. The entities are the nodes defining a class, hence an object having some properties, subclasses, relations and a key role into the ontology.
- the **Properties**, represented as directed arrows, defining details about the relationships between two or more entities.
- the **Individuals**, labeled with a violet rhombus. The individuals are, instead, nodes representing instances of a specific entities: the biological gender can be male or female, the eye colour can be brown, green, blue.

For this reason, it is smart to apply ontological approaches to the cybersecurity environment, a field that continues to evolve over the years and that is becoming increasingly more complex as technology and knowledge progress. For instance, Quantum Computing is one of the most important topics of interest for several countries, opening the door to new opportunities in both IT and the cybersecurity domains.

Ontologies can represent the cybersecurity knowledge in a structured and semantic fashion by defining the relations between assets, threats, vulnerabilities and cybersecurity controls, easing the assessment process. Thus, the construction of an ontology provides a knowledge model capable of describing IT systems and other complex systems, especially in the context of companies, where their complexity often makes difficult to infer interdependent threats and link assets, making hard to detect hidden threats or data correlation.

In a nutshell, we can define the ontology from an high level as the main representation of a model having entities with relationships between them, and so defining what are the main components of the system from an abstract point of view, in order to visualize the potential connections between the several nodes of a system.

Instead, we refer to *sub-ontologies*, when applying an ontology to a specific system and using the defined inference rules on that system. As an example of application, we can consider the cybersecurity ontology, which links asset entities with vulnerability entities, thereby inferring the existence of a threat. Consequently, whenever a new asset is added to the sub-ontology and a new vulnerability is linked to that asset, we can infer that a new threat exists, which may lead to potential attacks from external or internal vectors.

The usage of an ontology brings very interesting features, since having a structured model to represent the elements involved in the risk assessment process makes possible to automatize the creation of inference rules, allowing to delete or partly remove the human error and the misinterpretation of the information by experts, that only relies on their experience and knowledge. Moreover, having ontologies leads to very important advantages on the efficiency of the system representation due to the possibility to dynamically and promptly be updated and evolved whenever new threats or vulnerabilities are found. In fact, the object-oriented representation of entities offers the advantage of allowing the system to grow both vertically and horizontally, incorporating additional relationships between nodes. However, although the ontological approach is recommended for the reasons discussed above, its downside is the complexity encountered when creating a new ontology. It is not trivial to organize the domain's elements and avoid errors in complex systems. Moreover, the ontology must be updated to align with emerging threats and

remain consistent with the state of the art, which requires significant effort from the operators. For this reason, ontologies are usually not built from scratch; rather, they are developed by building on existing, even if incomplete or partially developed, ontologies, making it easier for authors to create coherent sub-ontologies tailored to specific cases or studies.

2.3 Threat Propagation and Attack Graph

As briefly discussed in section 2.1, there are several cybersecurity risk assessment approaches, that differ in the way the assessment is performed, the starting and the intermediate points, and sometimes the metrics that are used to achieve the final goal.

However, another aspect that must be highlighted and that it is not fully covered by the cybersecurity risk assessment nowadays is how the threats in a node of the IT environment can *propagate* over the system and how it can affect the evaluation of the risks. In fact, the cybersecurity risk assessment performs an evaluation regarding the so called "well-known vulnerabilities", even if nowadays automatic tools are able to significantly help the assessment process, by founding the vulnerabilities and the potential threats in the system, supported by many open technologies that are accessible by any one in the web and that are widely used by the experts.

Companies and whatever entities involved in the cybersecurity assessment must be aware of the state of the art of the new attacks, that are not known a priori and so it is impossible to be assessed in a "static" way. Moreover, attackers may overpass cybersecurity controls and protections by exploiting the vulnerabilities in other nodes, performing complex and articulate attacks, that usually are composed by many steps before reaching the final goal. The sequence of the steps are commonly known as *attack patterns* and are, even in the past, used to try to predict the possible tactical layers and techniques involved by adversaries to create threat models to overcome the weakness of the basic cybersecurity risk assessment. The attacker can be driven to target a company asset by many reason, and understanding which are these reason is crucial for the interpretation of an attack pattern, along with the technique that is used to compromise the target (lateral movement, privilege escalation, and so on), that can give a lot of information considering which are the adjacent nodes that can be apparently be not related to the target node, but that rather opens the door to possible attack patterns.

Due to the complex IT systems and to the unstoppable evolution of the state of the art of the attacks, the analysis of the attack patterns is not trivial, considering the several amount of information to be analysed in an organization environment. Furthermore, the complexity of the possible scenarios does not ease the automatization of the processes involved into the attack patterns analysis and their application to the IT environments and no final solutions exist nowadays.

This document wants to go into the details of the threat propagation analysing the solutions in the literature, along with their advantages and disadvantages, in order to find a possible proposal leveraging on the automation of this process.

Several papers exist, of researchers that studied the possible solutions in the last tens of years and that gave their contribution to the deployment of proposal, even if only in

theory.

Attack graph is a widely used model for network-security analysis, that is able to analyse the attack paths that can be followed by an attacker in the described system, in order to achieve a specific goal. Understanding this will lead cybersecurity experts and companies to predict complex attacks and to apply cybersecurity countermeasures in a proactive way, highlighting which are the weaknesses of the system and at the same time being hand in hand with the state of the art and avoid catastrophic consequences.

As Zenitani [33] explains, an attack graph is composed by nodes describing the malicious event caused by the attacker and the edges correspond to the causal relations between events. Attack graphs can have many representations, and in particular the paper highlights the evolution of the latter. However, the main components are usually the same:

- **Network configuration**, namely the list of the hosts and the connection structure among them, including the software configuration on each host.
- **Vulnerability distribution**, hence the list of pairs of a vulnerability and a host to highlight the ones the host is affected by.
- **Threat**, namely whatever entity that is able to exploit a vulnerability to gain access to the system.
- **Attack template**, finally the description of the mechanisms and techniques the threats act to exploit the vulnerabilities to reach the target.

An attack graph merges all this information and creates a chain of exploits, the so called *attack path*. The attack path is not in general the smallest piece of the chain, as long as each chain can include smaller chain of exploits. In general, we can point out that the longer is an attack path, the more secure is the system, in the sense it is required more effort from an attacker that wants to compromise in such a manner the system.

Moreover, given the nature of an attack, the representation of the attack graph is usually described as a cyclic structure. The reason behind this sentence is that an attacker could need to come back to a previously compromised node (e.g. after having performed privilege escalation) to exploit a vulnerability, allowing the creation of cycles in the attack path.

The very first representation of an attack graph, early abandoned for the complexity of the generation algorithm, was the *state enumeration graph*. This type of graph needs a generation algorithm with exponential complexity: if we consider a network of $\mathcal{N}$ machines, each of them having $\mathcal{M}$ vulnerabilities, the graph will need a node for each possible state according to the machine and the vulnerability, namely 2^{NM} combinations. This situation clearly led to scalability problem even for small networks, and it is not applicable to real cases, as the first attempters Swiler et al. [31] and Ritchey and Ammann [26] highlight in their researches.

Figure 2.2 shows a perfect example of a very complex state-enumeration graph, that underlines the complexity of such network, taken from the paper written by Li et al. [17].

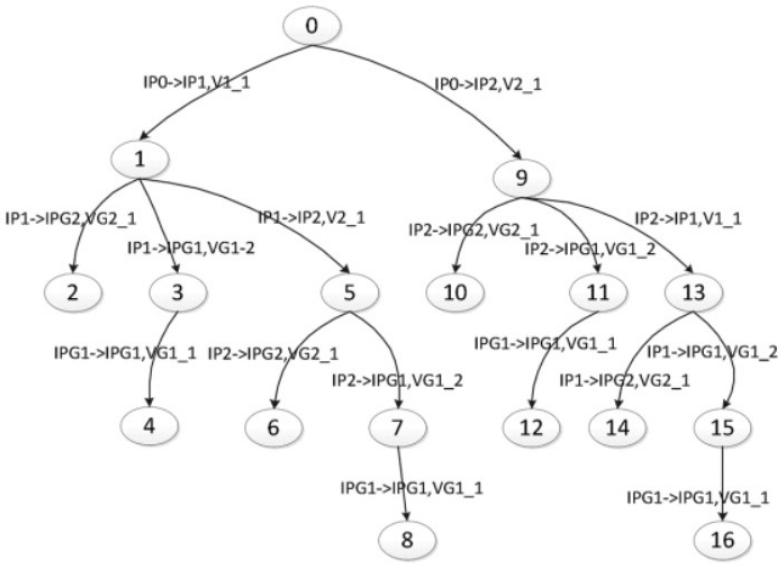

Figure 2.2. Example of state-enumeration graph [17]. We can observe the complexity of this kind of representation to be analyzed and traversed to infer the threat propagation

Finally, to overcome this problem, the *exploit-dependency graph* took hold in the subsequent papers. The concept paternity is to be given to Ammann et al. [1], while the node organization to Jajodia et al. [15]. This structure is a bipartite graph having two different types of node, and it is very important to understand this to go forward with this document. The graph includes the *exploit node* and the *condition node*, whereas the edges link the conditions to the exploits or vice versa. It is forbidden to have nodes of the same types linked together. With this configuration, it is additionally possible to differentiate the condition nodes into *preconditions* and *postconditions*. The preconditions are those conditions that are needed by a threat in order to make an exploit happens, hence they are necessary for that exploit. Rather, the postconditions are the fruits of the exploitation of a vulnerability or, more in general, of the success of the exploitation. In order to make a chain of exploits, the postconditions can be the precondition of other exploits. Given this macro categorization, we can further split the conditions into $c_{reachable}$, $c_{vulnerable}$ and $c_{compromised}$. The first two are possible preconditions, while the last one is the postcondition.

Figure 2.3 shows an explanatory example of the dependency-graph representation with the relevant nodes.

Since we can say an exploit node is reachable if and only if the preconditions are all satisfied, we can imagine the exploit node as an AND gate having the precondition nodes as inputs. The postconditions, instead, can have multiple parent exploit nodes, but actually it is enough for the attacker to reach only one exploit node, in order to gain the postcondition, namely we can interpret the condition node as an OR gate having as

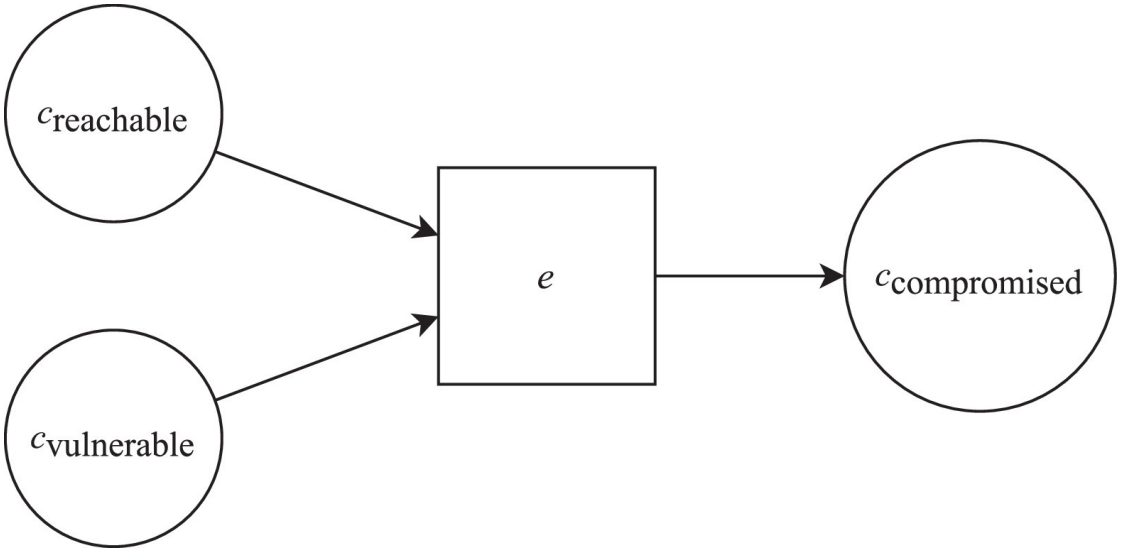

Figure 2.3. A *minimal attack graph* [33].

inputs the exploit nodes.

Once an overview about the attack graphs representation at an high level is given in this Section, we have the foundation of the threat propagation analysis, based, mainly on the exploit-dependency graph, which generation algorithm has a computational cost $O(N^2 + NM)$.

However, even if the representation of the attack graph is given, there is no specification about how to describe each node of the graph. In fact, the attack graph can be considered as an aggregation of the several attack paths involved during the analysis, so the representation of the attack templates.

2.4 MulVAL in a nutshell

One of the most important research in the field of attack graphs is the one conducted by Ou et al. [24], that proposed a Datalog-based open source solution, called MulVAL (Multi host, multi stage Vulnerability Analysis). Datalog is a logic programming language, subset of the Prolog language, that is used to create clauses which are used by the tool to gather information. Ceri et al [5] provided a full Datalog guide, describing how clauses can be built in a general context.

The main inputs to MulVAL are the listed below, followed by a brief description one by one and an example, taken from the paper:

- **Advisories**, such as the detected vulnerabilities.

- **Host Configuration**, in order to describe the list of software and services running on the host.
- **Network Configuration**, in order to describe the network devices' configuration (i.e. routers, firewalls, etc).
- **Principals**, namely the users of the network.
- **Interactions** between the components.
- **Policies**, to declare access permissions.

The Advisory is represented by a Datalog clause that describes the vulnerability that was detected in previous scans.

```
1    vulExists(webServer, 'CAN-2002-0392', httpd)
```

The Host Configuration can be retrieved by scanners as well, exporting information such as some parameters referred to a service program.

```
1 networkService(webServer, httpd, TCP, 80, apache)
```

The clause specifies that a *webserver* is running a program called *httpd* as user *apache* and listening at *port 80* using *TCP* protocol.

The Network Configuration is modelled as the host access-control list (HACL), that can be provided by network tools such as firewall management tools. An example of Prolog clause is the following.

```
1 hacl(internet, webServer, TCP, 80)
```

The clause specifies the incoming traffic from the *internet* to the *webserver* is allowed to flow to *port 80* with *TCP* protocol.

The Principal is a binding between a user and its account. Of course, each principal is related to specific policies, so it is important to specify this clause.

```
1 hasAccount(user, projectPC, userAccount)
2 hasAccount(sysAdmin, webServer, root)
```

The Interaction is crucial when we want to analyze possible attack paths. In fact, each stage of an attack can be influenced by some specific configuration, operating system or the vulnerability running on the specific node. The clause, in this case, is represented as a Prolog Horn clause, meaning that the first line stands for the Conclusion, while the following ones, preceded by the symbols ":-", are the Conditions.

```
1 execCode(Attacker, Host, Priv) :-
2     vulExists(Host, VulID, Program),
3     vulProperty(VulID, remoteExploit,
4         privEscalation),
5     networkService(Host, Program,
6         Protocol, Port, Priv),
7     netAccess(Attacker, Host, Protocol, Port),
8     malicious(Attacker)
```

This apparently hard to be understood clause is not actually complex, knowing how to read it. As the conditions must be verified, it is important to start from them, before reading the conclusion. If we have a potential *attacker* having access to a specific *host*, which is running a *program* on a specific *port* and *protocol* and with a specific *privilege* level, the attacker can exploit a vulnerability and execute arbitrary code on that host.

Finally, Policies describe which principal can have access to data and whit which permissions.

```
allow(Everyone, read, webPages)
allow(systemAdmin, write, webPages)
```

Taking in account the Horn clause, we can define some more points. The left-hand side is called *Head*, while the right-hand one is called *Body*. When an Horn clause has an empty body is called *Fact*, otherwise, it is called *Rule*. In this way, we can define *Interaction Rules* that can be used by the tool by its reasoning engine, avoiding the necessity to update them frequently, since they are as general as possible, characterizing general attack methodologies instead of specific ones.

The algorithm of this tool is divided into two main phases:

- the **Attack Simulation** stage, which analyzes all possible data accesses that can result from multistage, multihost attacks deriving them from the interaction rules
- the **Policy Checking** stage, in which the attack simulation output is analyzed and compared with the policies, in order to detect violations related to the permissions, and so outputting those violations.

However, it is important to point out that MulVAL is a flexible tool: even if this components are described, it can work even if it is not very complete (e.g. it can work without using any policy), but at the same time can be enriched with many other information, such as metrics, more specific interaction rules and so on. The options are described in the public project repository[7].

At the very end, providing all this files and inputs, the tool exploits its engine, the XSB Logic Engine [28], that is able to interact with databases and produce the attack graph into csv files, or in pdf format to visualize it, if you want, just embedding the tool called Graphviz [10], that is able to produce graphs.

The tool comes with two versions, so that it can be installed both in Linux-based systems or over other platforms. To have information on how to setup correctly the environment, Lin's Treasure [8] has written a very useful guide, even for the Docker version. Finally, the embedded graph generator is able to produce the output given the input files with a simple command.

```
graph_gen.sh input.P -v -p
```

[7] https://github.com/risksense/mulval

[8] https://kazumi.blog/posts/2023/07/how-to-setup-mulval/#mulval-from-scratch

2.5 Bayesian Network

The last section of this chapter is focused on the Bayesian Network (BN) and its properties. The attack graph alone is not able to provide the medium to analyse the attack paths: it allows to represent them, but it doesn't give to the user any information about the *probability* an attack is successful, that is the primary interest of experts when the attack pattern are studied.

Stephenson [30] wrote a paper to explain how a BN works, mixing together theory and usage. The BN is a peculiar structure that relies on a *Directed Acyclic Graph* (DAG). It means that, rather than allowing cycles and allowing bidirectional moves between nodes, it only allows to go from a node (called Parent or Ancestor node) to another one (the Child or Descendant) in just that direction, and, moreover, it does not allow the creation of cycles, but a child can't have more than one edge linked with the parent, that is an incoming one. Figure 2.4 and Figure 2.5 clarify the differences between the two, showing very simple examples to identify the differences.

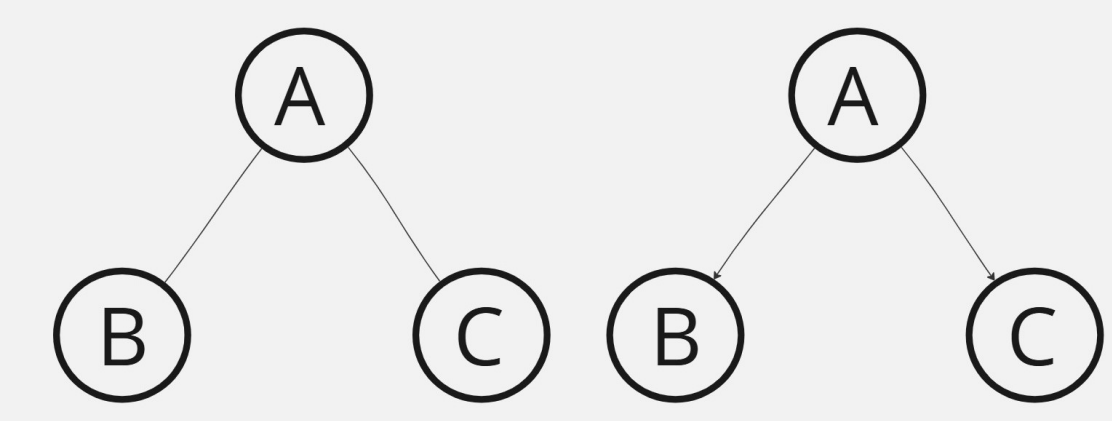

Figure 2.4. On the left an example of Graph model, with bidirectional edges. On the right, an example of Directed Acyclic Graph.

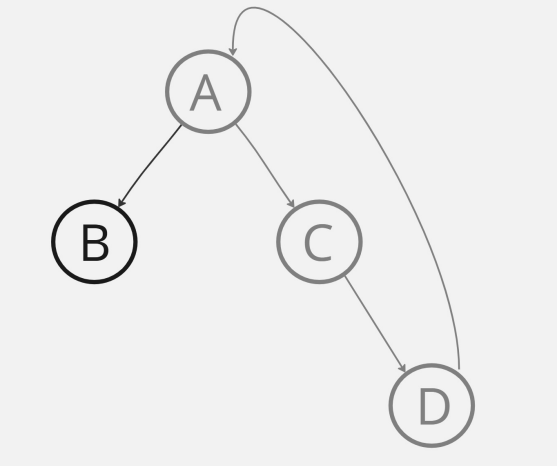

Figure 2.5. Example of graph with a cycle (the gray one). By definition, it cannot be considered as structure for a Bayesian Network, since it could allow to return back to a previous node.

We can represent a BN as a set of edges $E = \{e_1, e_2, e_3, ...\}$, each one connecting two nodes with the explained constraints, so respecting the DAG rules. If we consider two nodes, A and B, connected by an edge starting from A and ending at B, e_{AB}, it implies that node B is *conditionally dependent* on node A, meaning that the probability node B is *True* depends on the probability of node A is equal to *True*. In this case we speak about *Conditional Probability* and, accordingly to the paper, we speak about *Inference*. Inference is the task to calculate the probability of a node in a BN when the values of the other nodes are known. This property of the BN gives us the possibility to create *Conditional Probability Tables* (CPT) for each node, in order to be then inferred when we want to find a path. We can think the CPT as a table including some variables that must be evaluated, depending on the values of the ancestors, and so a table including the variables as rows and the ancestors' values as columns. The intersections will be the current variables' values in the specific node. The possible values are True or False, with their probabilities.

Figure 2.6 shows an example of Bayesian Network, including the Conditional Probability Tables.

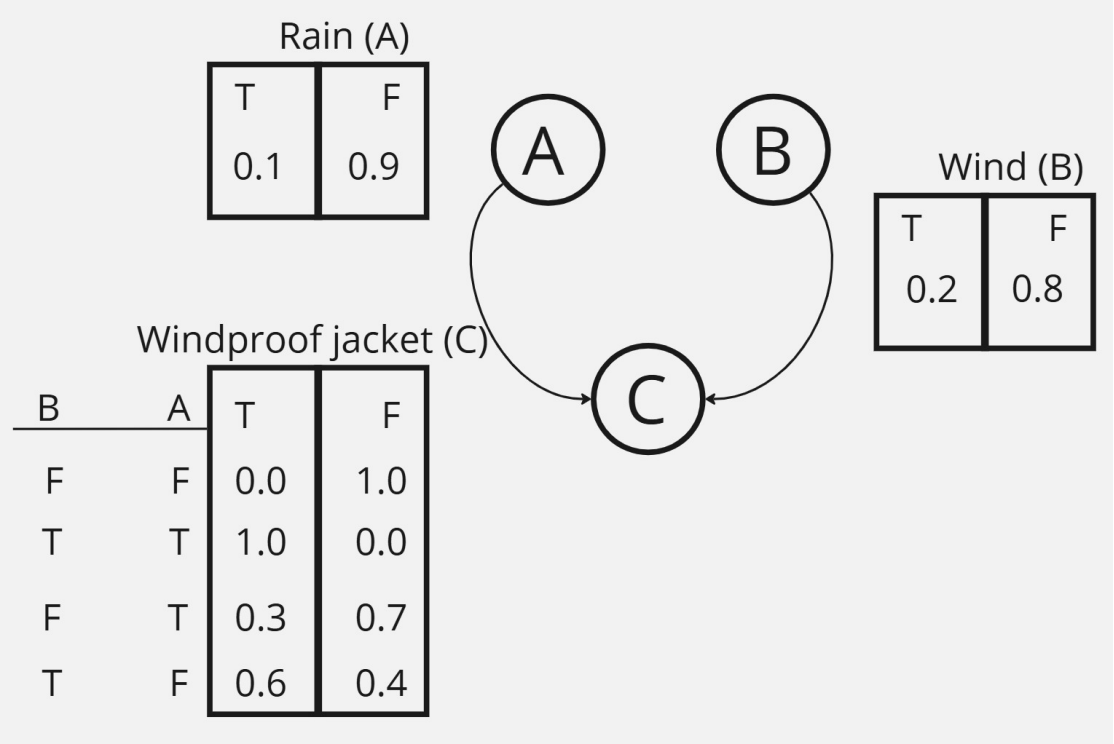

Figure 2.6. Each node of the Bayesian Network has its own CPT. Node C depends on the values of the ancestors A and B.

Creating the Bayesian Network implies knowing the local probabilities of each node, but it does not mean those probabilities are related each other, so they are not generated because of the connections between the nodes themselves. Rather, the conditional probabilities, called also *Marginal Probabilities* need to be computed via the inference in the BN. There is no a unique method to compute such probabilities, but they are many with their pro and cons. Depending on the necessities, we can distinguish two main categories:

- the **Exact Inference**, which computes the probabilities precisely, but they are computational expensive.
- the **Approximated Inference**, which rather computes the probabilities with faster algorithms, providing a degree of imprecision.

For the scope of this Thesis, this chapter will provide a description of only two methods belonging to the first category. In the field of security, it is important to achieve an high accuracy when calculating the probabilities as they are computed as quantitative as possible, but they are usually evaluated qualitatively; it is hard to quantify a probability, because it usually depends on several factors, strictly related to both technical or human

aspects. Even more, the probability diminishing we want to obtain can be meaningless if we don't give the right weight to the latter. To understand this sentence, I propose two toy questions: is it worth to spend more money to implement additional controls and achieve a 5% less probability of an attack to be successful? How much does the attacker effort increase if it wants to compromise a critical asset?

For this reason, the two methods that are summarized in Table 2.2, with advantages and disadvantages are the *Variable Elimination* and the *Belief Propagation*, which belong both to the category of Exact Inference. Actually, even hybrid solutions can be explored, since in complex systems and networks the number of nodes can be increasingly costly to be analyzed through a BN, with particular attention to enterprises.

Inference Method	**Description**	**Advantages**	**Disadvantages**
Variable Elimination (VE)	It simplifies the process of conditional probability calculation by eliminating the *hidden variables* from the computation.	Suitable for single queries.	Limited scalability, and it must be repeated for each query.
Belief Propagation (BP)	It propagates beliefs via messages through nodes, first with a bottom-up approach, then with a top-down one, to update the marginal probability.	Suitable for multiple queries, even if there are cycles.	It may converge very slowly or not at all in networks with loops.

Table 2.2. Advantages and Disadvantages of Variable Elimination and Belief Propagation

Chapter 3

Related Works

While Chapter 2 gives information about the cybersecurity risk assessment methodologies, the theory about what concerning the attack graph in the context of threat propagation and about the Bayesian network, along with the technology used to generate the attack graph itself, this chapter of the thesis will focus on the works that were useful to understand which is the state of the art of threat propagation and threat analysis, aligning them with the current used technologies and synergies on what discussed in the previous chapter. The information gathered from the following sources have been used to design a possible solution to automate the cyber attack simulation in IT infrastructures, presented in Chapter 4, with the goal to compute the attack paths probabilities and to predict the possible attacker behaviour, since they discuss frameworks that have been already validated with positive results.

3.1 IT Ontology Model

The papers by De Rosa et al. [9] and Maunero et al. [19] show how ontology can be applied in the field of cybersecurity, highlighting the bullet points and showing a project called ThreMA, that is placed at the foundation of the idea behind this document, since this work pretends to be a kind of continuation of such a project.

As the name suggests, the concept of threat propagation requires a thorough understanding of the threats that are present in an IT infrastructure. ThreMA is designed to support this understanding by automatically identifying and mapping these threats to specific ICT components, enabling analysis of possible propagation paths using a structured ontology.

ThreMA [9] is a threat management tool suitable for various ICT environments, working in coordination with a security team to provide insights into potential security risks. By leveraging an ontological model, ThreMA offers both scalability and adaptability, which are essential for handling complex and changing cybersecurity challenges.

In Figure 3.1, the architecture of ThreMA is shown, standing thanks to an *ontology-metamodel* divided into three sub-ontologies:

- **ICT sub-ontology**: Contains the vocabulary and rules required to model the ICT

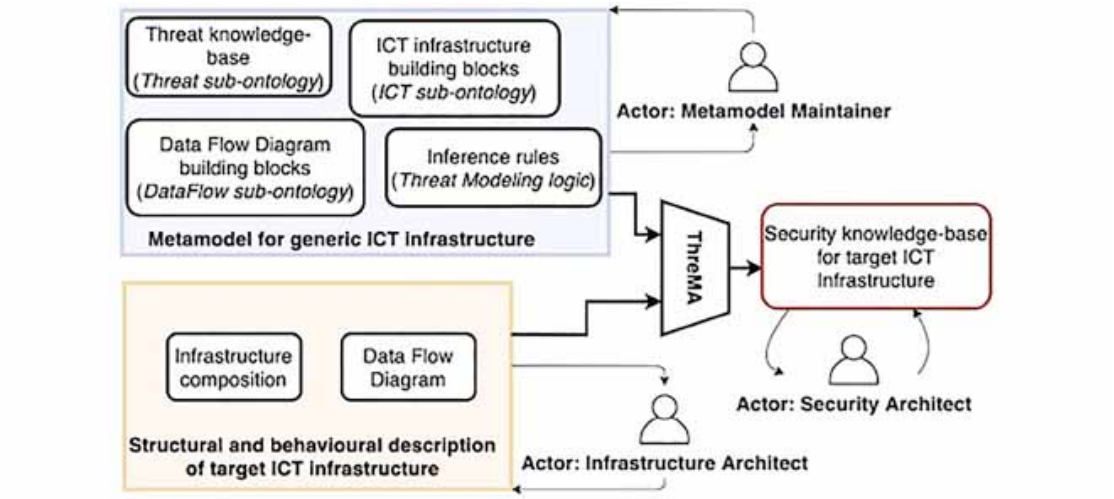

Figure 3.1. The ThreMA architecture [9].

infrastructure. This sub-ontology describes each component of the system's architecture, enabling detailed representation of assets and their relationships.

- **Data Flow sub-ontology**: Represents the data flows between components, enabling a traceable path for information exchanges across the system. This level of traceability is essential for identifying how threats could potentially propagate through the system.
- **Threat sub-ontology**: Models various types of threats based on their properties and potential behaviour patterns. This sub-ontology enables ThreMA to systematically map threat types and their relationships with system assets, supporting a structured approach to threat modelling.

ThreMA's ontology is developed using the Protégé [21] tool, with OWL2 [6] as the ontology description language, allowing for detailed and formal representations. To reason over the threats and ICT components, ThreMA utilizes SWRL (Semantic Web Rule Language) [14], specifying rules that allow for automated reasoning across the ontology. Through these inference rules, ThreMA automates the identification and classification of threats, providing the insights necessary for a proactive approach to cybersecurity. The inference engine applies these rules across the ontology's components to enable the detection of potential threat scenarios, which is fundamental for understanding and predicting how threats may propagate.

The paper by De Rosa et al. [8] introduces a framework that supports cybersecurity governance through a dedicated ontology for ICT systems. The objective of this work is to provide a structured and semantically rich model to manage and secure complex ICT

infrastructures, where the relationships among various assets and security requirements demand an organized and adaptable approach.

The developed ontology enables a systematic representation of key ICT system components, including assets, actors, vulnerabilities, and threat vectors. This structured model facilitates the identification of security gaps and interdependencies that may impact the overall security of the system. One of the primary features of this model is its integration of risk assessment within the governance structure, explicitly representing threats and vulnerabilities. This setup allows for a dynamic risk evaluation, supporting security teams in proactive responses as the threat landscape evolves. The vulnerability and attack ontology represented in this paper is shown in Figure 3.2 to give an abstraction of the concept behind this work.

Figure 3.2. Vulnerability and attack ontology [8].

Additionally, the ontology aims to automate various aspects of cybersecurity governance by leveraging inference engines for tasks like compliance verification, risk prioritization, and policy enforcement, thus reducing human error and improving the efficiency of security management. The authors include real-world scenarios and case studies to demonstrate how this model can be applied across different ICT contexts, confirming the ontology's effectiveness in capturing the complexities of cybersecurity governance.

Building on the foundations of ThreMA, the paper by Maunero et al [19] takes a step further, proposing a method to automate the entire cybersecurity risk assessment process. While ThreMA is focused on threat modelling in ICT infrastructures, this approach expands the use of ontologies to support risk assessment, from threat identification to the evaluation of possible impacts.

This framework uses ontologies not only for threat detection but also for risk analysis, taking into account aspects such as asset criticality, likelihood of threats, and potential

impacts. This additional detail allows for an analysis that not only models threats but also prioritizes them based on risk levels, facilitating the identification of high-risk paths in the infrastructure.

The ontological framework proposed by Maunero et al. is structured to support reasoning that goes beyond ThreMA's basic threat mapping, offering:

- **Extended Threat Modeling**: Beyond detecting threats, this approach incorporates attributes like vulnerability severity and asset exposure, allowing for a more comprehensive assessment of threat propagation paths.
- **Automated Risk Scoring**: Each threat is assigned a risk score based on its impact and likelihood, enabling the system to prioritize threat responses. This scoring system supports more precise mitigation strategies by focusing on high-priority threats.
- **Propagation Analysis**: The ontology allows for simulations of potential threat propagation scenarios across interconnected assets, evaluating the likelihood of different propagation paths within the infrastructure.

This approach creates a complete risk assessment model that automates threat and risk management processes. The ontological methodology aligns with the objectives of this document, which aims to implement threat propagation analysis to not only detect threats but also assess their potential impact on system security. Expanding on the principles established in ThreMA, this work provides a foundation for a cybersecurity assessment framework capable of adapting to evolving threats through automated, ontology-based reasoning. This ontology serves as a basis for implementing threat propagation analysis, enabling the systematic prediction and response to complex threat scenarios.

Finally, the paper written by Fenz Stefan [11] bridges the conceptual clarity of ontology with the inferential power of Bayesian networks to address the complex challenge of threat probability estimation in cybersecurity. By leveraging ontology, developed based on the guidelines provided by the NIST SP 800-12 [22], the work introduces a structured model of ICT systems, integrating assets, data flows, and associated threats into a coherent framework. The ontology not only defines the relationships between these entities but also serves as the foundation for reasoning about their interactions and potential vulnerabilities. What sets this approach apart is the seamless incorporation of Bayesian Networks into the ontology-driven model. While the ontology defines the semantic structure, the Bayesian network adds a probabilistic dimension to the framework, enabling dynamic threat propagation analysis. The probabilistic model allows to assess how specific vulnerabilities or compromises in certain nodes (e.g., assets) can lead to cascading effects throughout the system, thereby quantifying the likelihood of an attack reaching critical assets. This approach exemplifies the potential of combining declarative and probabilistic methods to address cybersecurity challenges. It builds on existing works that use ontology for threat modelling, such as ThreMA, but innovates by linking this formalism to probabilistic reasoning. The result is a framework capable of handling both the qualitative understanding of threats and the quantitative estimation of their impacts. In fact, even if it is now impossible to work on standard metrics and measures allowing experts to use

quantitative approached to the evaluation of the risk, it is in any case possible by an enterprise to switch from a qualitative approach to a quantitative one, following internal rules and policies. However, the work also highlights the challenges associated with integrating these two paradigms, particularly in managing the scalability of Bayesian networks as the system complexity grows. Overall, the paper underscores the value of unifying ontology-based models with Bayesian reasoning to provide a more holistic approach to threat assessment. This methodology lays the groundwork for an automated system that not only models ICT infrastructures but also predicts the likelihood of specific threats materializing, offering a valuable tool for risk management and decision-making.

3.2 Attack Graph and Bayesian Network

Along with the work by Zenitani described in Section 2.3, used as explanatory guide by the latter in order to create attack graph and considered as milestone for the same, other papers were deeply analysed, in order to better understand the literature behind the threat propagation and the state of the art.

The following works still include concepts and ideas that rely upon Ontolgy and Bayesian Networks, even if in a different way, but at the same time highlighting the potentiality of the model based on that kind of structure, that opens the door to several researches and possible implementations. Since the Thesis speaks about MulVAL in the previous chapter, it won't be mentioned in this one, even if it can be considered as a point of departure for many other paper.

D'Ambrosio et al. [7] heavily relies on the paper written by Zenitani and on MulVAL to show a possible implementation of the attack graph using *Bayesian Networks*. The authors propose a Bayesian Threat Graph model that represents potential threat propagation paths in a system, allowing for a structured analysis of both external and internal threats. This approach demonstrates how Bayesian networks can model dependencies and conditional probabilities associated with various attack paths, offering a probabilistic perspective on risk exposure. A core focus of the work is on capturing the unique risk dynamics introduced by insider threats—an area often overlooked in traditional attack modelling. By quantifying insider threat probabilities and incorporating them into a broader threat graph, the paper enables a more comprehensive risk assessment. This methodology is particularly relevant for systems where insider actions can directly impact security, as it offers a means of identifying and prioritizing high-risk scenarios based on probability-based threat propagation.

The paper written by Tong Li et al. [18] highlights the efficiency and practical value of leveraging attack patterns in security analysis, demonstrating how these structured patterns can significantly enhance threat identification and response. By using attack patterns, the authors show that security professionals can proactively anticipate various attack vectors and apply pre-defined strategies to counter specific types of vulnerabilities, improving both the accuracy and speed of threat modelling. One of the core concepts presented is that attack patterns serve as a bridge between known vulnerabilities and the tactics used to exploit them, thereby aiding in vulnerability assessment and risk management. The paper demonstrates how attack patterns can be mapped to specific threat

actions, revealing the underlying mechanics of various attacks. This mapping allows for a more comprehensive view of how certain vulnerabilities may be exploited across different scenarios, making it easier to predict and mitigate similar threats in the future. However, the study also underscores a critical limitation: the manual effort required to build and maintain the attack models developed in the paper is substantial, making this approach less feasible for real-world, large-scale systems. The process of manually constructing and updating these models is time-consuming and complex, which presents a significant barrier to implementing such methodologies on a broader scale. This limitation indicates a gap between the theoretical effectiveness of attack patterns and their practical application, suggesting a need for automation or alternative methods to make these models sustainable in a dynamic threat landscape.

The paper by Hankin et al. [13] presents a framework for automatically generating attack graphs by combining system topology, CAPEC attack patterns, CWE weaknesses, and CVE vulnerabilities. The approach focuses on representing potential attack paths within a network by integrating information about assets, vulnerabilities, and exploit conditions. The framework models attacks using nodes and relationships, where exploit nodes represent specific actions taken by attackers (e.g., exploiting a vulnerability) and condition nodes represent system states required for an attack to progress. This combination allows the dynamic generation of graphs that trace possible pathways attackers could use to compromise a system. By integrating threat databases and automating the process, the framework emphasizes efficiency and scalability in generating attack scenarios, making it a significant step toward automated cybersecurity risk analysis.

The idea to use frameworks like CAPEC to map attack patterns is valuable, but however it requires too much technological effort in order to achieve the expected result. In fact, nowadays there are no ways to easily link CAPEC patterns to the CVEs, but a link to the CWEs must be found first, and that link not always exist. So, at the very end, the possibility to use CAPEC as main instrument to create attack graph was discarded, even if other works talk about it as a possible solution for future works, since a common evaluation scale to evaluate threats and vulnerability in a quantitative way is always more needed nowadays and, for this reason, the frameworks are growing up during the years and many researchers are collaborating in such a way, so future integration are taken into account in the literature.

Chapter 4

Contribution

The previous chapters allow the reader to understand which are the technologies, tools and mathematical theories that are applied into the work described into this Thesis, giving a general vision of the current literature in the field of cybersecurity risk assessment and threat propagation, highlighting the major works done in what can be an attempt to automatize a very complex process that nowadays is mostly done manually and with no agreement between experts about which metrics to use and which frameworks to apply into the assessment.

This Thesis claims to propose a methodology that can partly overcome the non-automation problem, achieving good results basing on the papers presented in Chapter 3 and combining them, taking advantage from frameworks and tools that are used, but that are not integrated each other to ease still mechanical and hard operations. The full code can be found into the Appendix **??** and into my GitHub [1].

The idea behind the semi-automation of the threat propagation analysis process is to split the pipeline into three main steps:

1. create the **Cybersecurity IT Ontology** to represent the enterprise network infrastructure and characteristics of the nodes.

2. give the adapted ontology to **MulVAL**, in order to generate the attack graph.

3. take the MulVAL's output and the suitable metrics to generate an **Attack Bayesian Network** and compute the attack paths probabilities.

Figure 4.1 shows graphically which the pipeline is composed of.

The remaining of this chapter aims to expand each step of the proposed solution and to give some more details about how it can be implemented. In particular, the focus of this chapter will point out the third step, since the Thesis provide a Proof-of-Concept realized with Python [32], that will be further shown, with its classes and main methods, relying in modules such as Pandas [20], NetworkX [12], PgmPy [2] and Arango [23].

[1] https://github.com/StefanoSIIacono/TP-Thesis

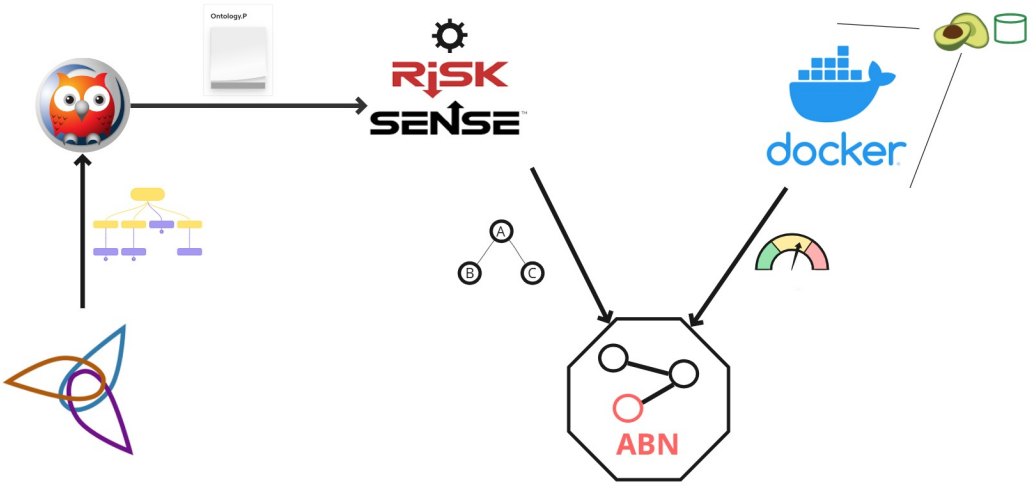

Figure 4.1. The proposed pipeline. The first step is represented by the ontology created with tools like Protégé, to represent the IT infrastructure. The ontology is than parsed to Prolog, passed to MulVAL, that runs the reasoning engine to create the attack graph, used, along with the CVEs weights taken from ArangoDB, to generate the ABN, with the tool provided in this thesis.

4.1 Ontology-based Infrastructure

The first step of the pipeline is crucial for understanding and describing the infrastructure and topology of an enterprise's network as comprehensively as possible. This means that every logical and physical component that could potentially contribute to the compromise of business continuity must be mapped within the Cybersecurity IT Ontology.

To achieve this, tools like WebProtégé [2] can be used to create classes and represent each node of the topology. The core idea is to model all assets, vulnerabilities, threats, controls, data flows, and other cybersecurity-related elements as *Objects* and to establish relationships between them using *Properties*. Each class, that is the high visualization of the instances, includes a certain number of *Individuals*, namely the real instances that model the IT infrastructure under analysis. This approach follows a logical structure that allows for future updates to the ontology. As mentioned in previous chapters, the strength of an ontology lies in its flexibility and adaptability, enabling it to stay up to date with state-of-the-art developments, including zero-day vulnerabilities and emerging threats.

[2] https://webprotege.stanford.edu/

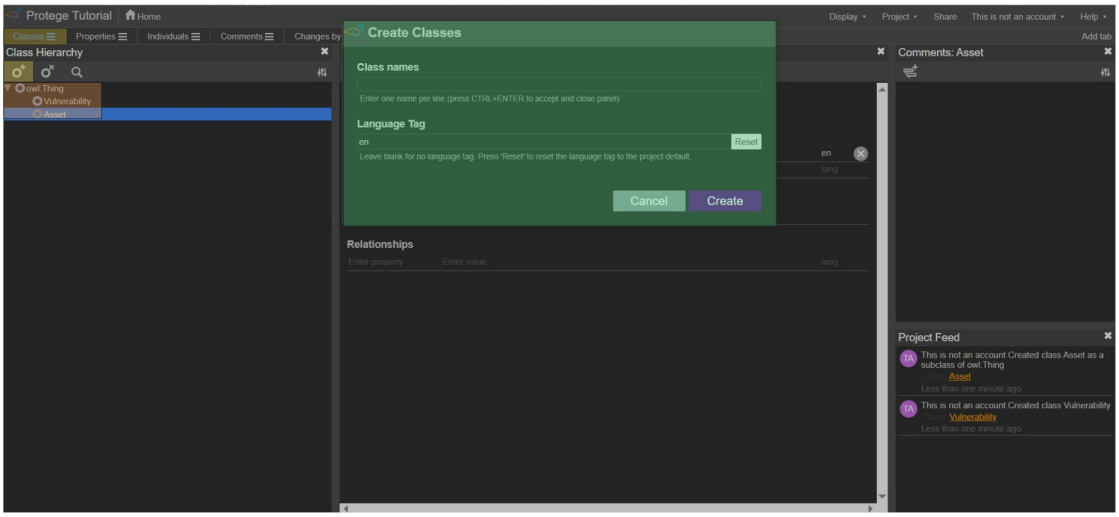

Figure 4.2. The WebProtégé Dashboard. In section Classes it is possible to create new instances with the button highlighted in yellow. The list of the created classes is highlighted in orange.

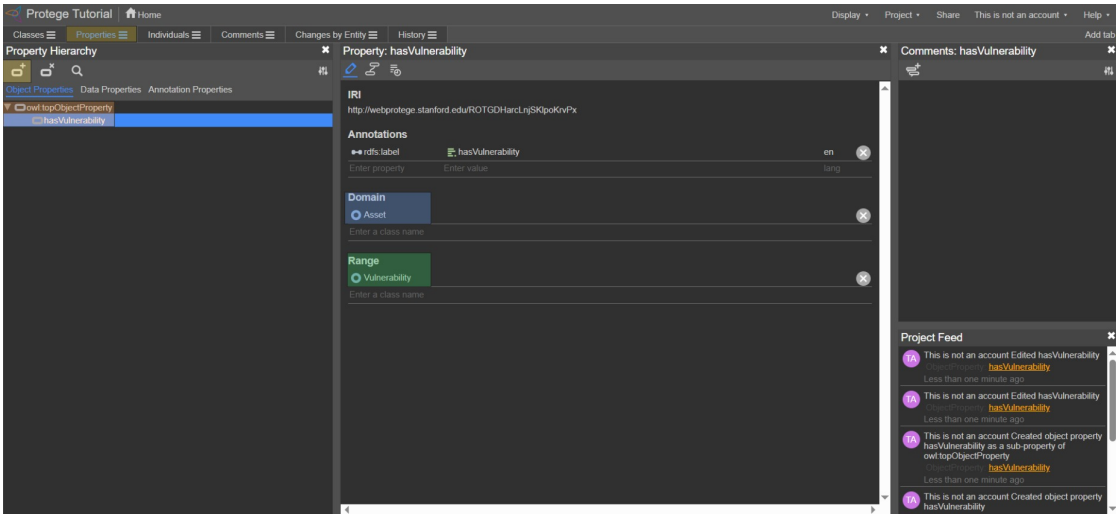

Figure 4.3. In section Properties it is possible to create the links between the classes.

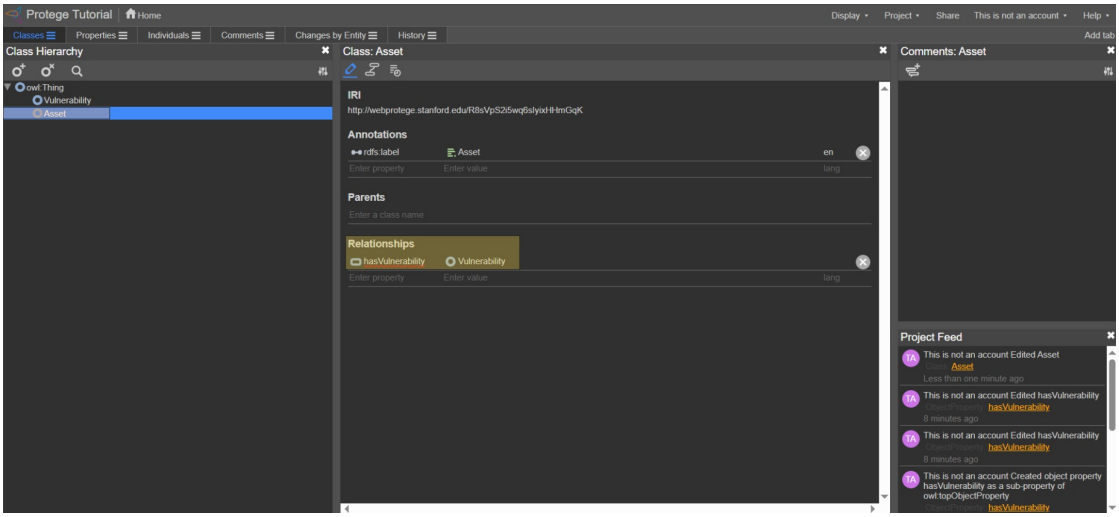

Figure 4.4. It is possible to select the relationships a class owns, accordingly to the logic of the ontology.

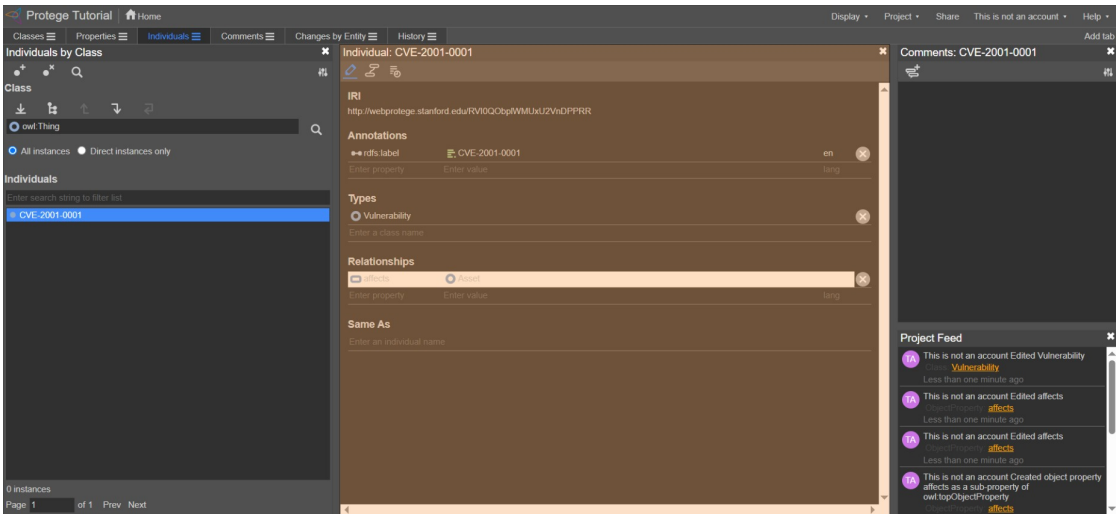

Figure 4.5. The individual creation. It is possible to specify the important parameters, given by the WebProtégé GUI.

With WebProtégé it is possible to create complex ontologies, represented as complex networks of classes. The main actions it is possible to perform on WebProtégé are the

following:

- Class creation, inserting the name of the class (Figure 4.2).
- Property creation, inserting the name of the property, along with the Domain, hence the source class, and the Range, namely the destination Class (Figure 4.3).
- Relationships creation (Figure 4.4).
- Individual creation, inserting the class it belongs to and the relations it is related to (Figure 4.5).

As a starting point, however, the thesis builds upon a project that is, in a way, related to this one, namely ThreMA, previously described in Section 3.1. This project is already quite complete and makes use of frameworks that are expected to expand further in the future. Moreover, despite its current complexity, it can be easily updated. For these reasons, the ontology proposed in that framework perfectly fits the content of this thesis, even cause it is common to take an already existing ontology to be updated and enriched.

Nevertheless, as mentioned before, the issue with MulVAL is that it currently only accepts Prolog input files, whereas ontologies typically use different languages to represent their instances, such as OWL2. To figure out it, the proposed solution designs an *Ontology-to-Prolog* converter, capable of systematically transform an individual from the ontology into a Prolog clause that describes that component of the topology. Since the ontology is complete, it only requires a mapping from its classes to the clauses that MulVAL can interpret in order to generate an attack graph, so clauses describing the assets, the hosts, the network configuration, the vulnerabilities and so on.

Additionally, ThreMA is capable of scanning the network to identify vulnerabilities and threats affecting assets. This feature can also be leveraged to map vulnerabilities into Prolog, further enhancing the integration between the ontology and MulVAL.

4.2 From MulVAL to the Attack Bayesian Network

The second and third steps are closely related to each other, to the point that they can be merged into one. In fact, when the inputs are provided, MulVAL is able to automatically generate the attack graph in the form of csv files, or even graphically into a pdf file, and, moreover, it also gives the *running rules*, namely the subset of the rules included in *interaction_rules.P* and affecting the graph generation, that are applied to the specific case defined in the input file. This rules will be further integrated into the Attack Bayesian Network, since they also gives information about the probability of the nodes and they are fundamental for the conditional probability tables building.

The interaction_rules.P file is the default one, also used in this thesis, even if it can be customized by the user, since MulVAL is also able to be provided of metrics to be used in the graph generation. It defines how entities (e.g., hosts, programs, users) interact in a cybersecurity context. These rules are used to model the dynamics of a cyber attack, specifying how vulnerabilities, permissions, and network configurations can lead to compromises or unauthorized access. Into this file, the primitive and derived predicates are

declared, followed by the tabled derived predicates and the interaction rules. Figure 4.6 and Figure 4.7 show the list of predicates the interaction_rules.P file is made of.

```
/*************************************************/
/****         Predicates Declaration             *****/
/*************************************************/

primitive(inCompetent(_principal)).
primitive(competent(_principal)).
primitive(clientProgram(_host, _programname)).
primitive(vulExists(_host, _vulID, _program)).
primitive(vulProperty(_vulID, _range, _consequence)).
primitive(hacl(_src, _dst, _prot, _port)).
primitive(attackerLocated(_host)).
primitive(hasAccount(_principal, _host, _account)).
primitive(networkServiceInfo(_host, _program, _protocol, _port, _user)).
primitive(setuidProgramInfo(_host, _program, _owner)).
primitive(nfsExportInfo(_server, _path, _access, _client)).
primitive(nfsMounted(_client, _clientpath, _server, _serverpath, _access)).
primitive(localFileProtection(_host, _user, _access, _path)).
primitive(dependsOn(_h, _program, _library)).
primitive(installed(_h, _program)).
primitive(bugHyp(_,_,_,_)).
primitive(vulExists(_machine,_vulID,_program,_range,_consequence)).
primitive(canAccessFile(_host, _user, _access, _path)).
primitive(isWebServer(_host)).
meta(cvss(_vulID, _ac)).
```

Figure 4.6. The primitive predicates included into the interaction rules file.

```
derived(execCode(_host, _user)).
derived(netAccess(_machine,_protocol,_port)).
derived(canAccessHost(_host)).
derived(accessFile(_machine,_access,_filepath)).
derived(accessMaliciousInput(_host, _principal, _program)).
derived(principalCompromised(_victim)).
derived(dos(_host)).
derived(logInService(_host, _protocol, _port)).

meta(attackGoal(_)).
meta(advances(_, _)).

/*************************************************/
/****         Tabling Predicates              *****/
/*   All derived predicates should be tabled       */
/*************************************************/

:- table execCode/2.
:- table netAccess/3.
:- table canAccessHost/1.
:- table canAccessFile/4.
:- table accessFile/3.
:- table principalCompromised/1.
:- table vulExists/5.
:- table logInService/3.
```

Figure 4.7. The derived predicates included into the interaction rules file.

Below, we break down the key sections and rules in the file. Since the list is very long, only some instances will be explained.

The **predicates** define the basic entities and relationships in the model. For example:

```
primitive(vulExists(Host, VulID, Program)).
```

This indicates that a vulnerability (`VulID`) exists on a program (`Program`) on a host (`Host`).

```
primitive(hacl(Src, Dst, Prot, Port)).
```

This defines a network access rule, where a host (`Src`) can access another host (`Dst`) via a protocol (`Prot`) and port (`Port`).

```
primitive(attackerLocated(Host)).
```

This specifies that the attacker is located on a particular host.

```
primitive(hasAccount(Principal, Host, Account)).
```

This indicates that a user (`Principal`) has an account on a host with specific permissions (`Account`).

Rather, **tabling** is used to optimize rule evaluation by storing intermediate results and avoiding redundant calculations, such as

```
:- table execCode/2.
```

The **interaction rules** describe how security conditions (e.g., vulnerabilities, permissions) can lead to compromises or unauthorized access. Below are some of the most significant rules. There several types of interaction rules. For example, the Code Execution (`execCode`), the Network Access (`netAccess`), the File Access (`accessFile`), the User Compromise (`principalCompromised`), the Vulnerabilities (`vulExists`), the Malicious Input Access (`accessMaliciousInput`). Some example is listed below.

```
execCode(Host, Perm) :- hasAccount(Principal, Host, Perm).
```

If a user (`Principal`) has an account on a host (`Host`) with certain permissions (`Perm`), they can execute code on that host.

```
execCode(Host, Perm) :- principalCompromised(Victim), hasAccount(Victim,
    Host, Perm).
```

If a user (`Victim`) is compromised and has an account on a host (`Host`), they can execute code on that host.

```
execCode(Host, root) :- vulExists(Host, _, Software, localExploit,
    privEscalation).
```

If a local vulnerability (`localExploit`) exists on a host, a user with permissions can execute code as root.

```
netAccess(H2, Protocol, Port) :- execCode(H1, _Perm), hacl(H1, H2,
    Protocol, Port).
```

If a host (`H1`) can execute code and has access to another host (`H2`) via a network rule (`hacl`), it can access `H2`.

```
netAccess(H, Protocol, Port) :- attackerLocated(Zone), hacl(Zone, H,
    Protocol, Port).
```

If the attacker is located in a zone (`Zone`) and there is a network access rule (`hacl`) between `Zone` and a host (`H`), the attacker can access `H`.

```
accessFile(Host, Access, Path) :- execCode(Host, User), canAccessFile(
    Host, User, Access, Path).
```

If a user can execute code on a host (`execCode`) and has access to a file (`canAccessFile`), they can access that file.

```
principalCompromised(Victim) :- hasAccount(Victim, Host, _Perm),
    execCode(Host, root).
```

If a user (`Victim`) has an account on a host (`Host`) and the host is compromised (`execCode` as root), the user is compromised.

```
vulExists(Host, ID, Software, Range, Consequence) :-
    vulExists(Host, ID, Library, Range, Consequence), dependsOn(Host,
        Software, Library).
```

If a vulnerability exists in a library (`Library`) and a program (`Software`) depends on that library, the program is vulnerable.

```
accessMaliciousInput(Host, Victim, Software) :- inCompetent(Victim),
    hacl(Host, MaliciousMachine, httpProtocol, httpPort),
    attackerLocated(MaliciousMachine).
```

If an inexperienced user (`inCompetent`) browses a malicious website (`attackerLocated`), they may be exposed to malicious input.

Figure 4.8 shows the beginning of the interaction rules section.

```
/*****************************************************/
/****        Interaction Rules                 *****/
/*****************************************************/

/****** Section execCode ******
interaction_rule(
   (execCode(H, Perm) :-
        hasAccount(P, H, Perm)),
   rule_desc('Insider threat', 1)).
*/

interaction_rule(
   (execCode(Host, Perm) :-
        principalCompromised(Victim),
        hasAccount(Victim, Host, Perm),
        canAccessHost(Host)),
   rule_desc('When a principal is compromised any machine he has an account on will also be compromised',
   0.5)).

interaction_rule(
  (execCode(Host, root) :-
        execCode(Host, _Perm2),
        vulExists(Host, _, Software, localExploit, privEscalation)),
  rule_desc('local exploit',
  1.0)).
```

Figure 4.8. The first portion of the interaction rules into the file interaction_rules.P. The full content of the file can be found in Appendix **??**.

The file, hence, describes the general interaction rules that the user is interested in for its test case, giving to the tool's engine the logic and the inference rules for the creation of the attack graph. In fact, the interaction rules declared here are based on the predicates, but they will be also used to generate the *running_rules.P* file, that is the list of the more general used rules taken from the first file and applied, coherently to the test case.

Consider now the file listing the nodes called VERTICES.CSV, in which each record describes a specific node of the attack graph, giving some information:

- the number
- the label, related to the interaction rule
- the node type, namely if it is a LEAF node, an OR node or an AND node
- the initial value

This file is crucial to re-build the attack graph and, more important, to compute the CPDs. The other file is the list of the edges, called ARCS.CSV. Its records describe the arcs of the attack graph, but it follows a different logic with respect to the classic "parent-to-child" relationship. Otherwise, the file contains records describing this type of relation: "precondition-to-postcondition". So, in a way, it is like a "child-to-parent" connection. This behaviour is indicated with a "-1", following the nodes belonging to the edge, like describing a negative weight.

As mentioned before, the nodes can be of three types, LEAF, AND or OR. The node type is used for the CPT computation, since it defines the methodology it is built, accordingly to the Zenitani's work. In fact, a LEAF node can be a starting precondition or a statement defining the existence of a vulnerability *vulExist*, and so the probability must be taken from somewhere and cannot be computed, since this nodes have no parents. The AND and OR nodes, rather, have parents, that can be of any type, and so their probability is affected by the probability of the parent nodes, so that the computation of the conditional probability depends on that factors. This distinction is crucial because it makes harder to structure the Bayesian Network, since each node can have zero parents, one parent only, as well as two or more parent nodes, and of course it makes the probability computation more dynamic.

This phase of the pipeline involves the usage of the tool object of this thesis, that is structured in this way:

- **main.py** is the main file, where the user must select the input file. It runs a minimal GUI with Tkinter, that opens dialog windows to select the correct files.
- **attack_graph.py** is the file including the *AttackGraph* class, that contains the attack graph variable, provided by NetworkX, the list of nodes and the list of edges. It provides the following methods
  - *load_data* loads the vertices and the arcs from the provided files, storing them into the Dataframes that Pandas provides.
  - *build_graph* adds the nodes and the edges to the attack graph.
  - *get_node_info* to get the information of a specific node.
- **bayesian_network.py** that fully relies in PgmPy, includes the class *SecurityBayesianNetwork*, that builds the ABN. It provides the following methods
  - *build_network* converts the attack graph into the ABN, also validating the inputs. In fact, if there is any inconsistency, it allows to arise a warning.

  - *_create_node_cpd* is the method that wraps the class building the CPD (Conditional Probability Distribution) tables.
  - *perform_inference* that performs the probabilistic inference on the network.

- **probability_calculator.py** that contains all the classes involved in the CPD creation, namely the *MulValRuleExtractor*, that extracts the probability for the LEAF nodes in case it is not a vulExist node, the *CVSSCalculator*, that manages the CVSS scores, and the *ProbabiltyCalculator*, that is the main calculator managing the CPD tables building. Further explanation will be provided about their methods.
- **arango_client.py** includes the *ArangoClient* class that allows to query the database to retrieve the information about the CVEs needed for the Bayesian Network.

Once the attack graph is created, all the information can be used to produce an ABN. However, as said before, the LEAF nodes' probabilities cannot be computed, but they have to be taken from somewhere. In order to figure out it, I decided to integrate a database that collects all the information concerning the threats data from the major frameworks in the cybersecurity fields.

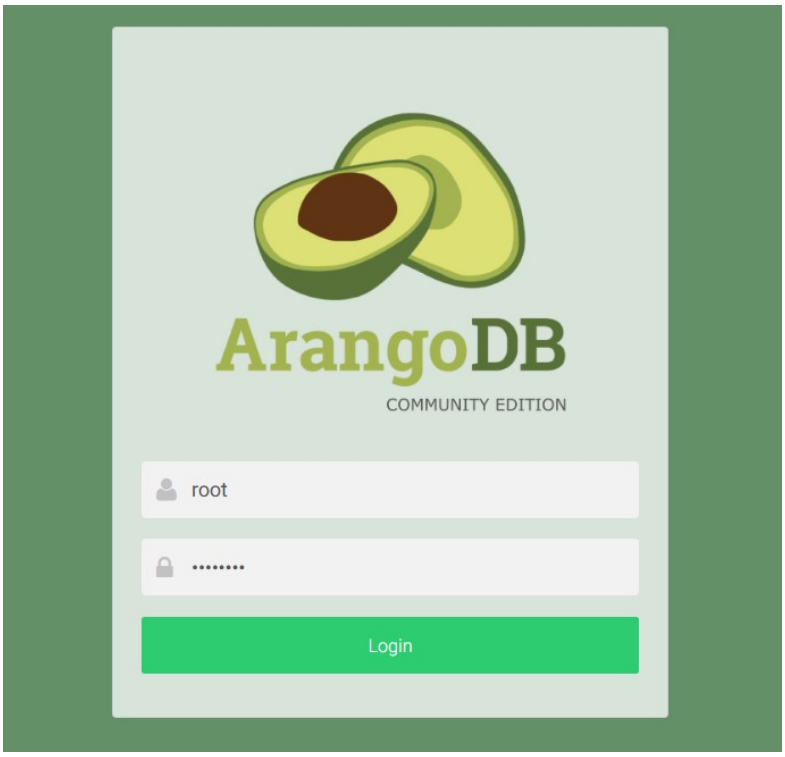

Figure 4.9. The BRON login page.

BRON [3] uses ArangoDB [3] as engine, since it supports graph, document, and key/-value data models, increasing performances, scalability and flexibility, to link together information taken from MITRE and NVD with bidirectional edges, so that it can be more flexible to be queried and easy to keep up to date. For this reason, it has the potential to grow along with the major frameworks and can be embedded in this project, since even this project claims to be improved in the future. The database can be easily installed via Docker, building the containers as shown in Figure 4.10, and exposing a service at localhost:8529 that will provide the dashboard as in Figure 4.9, giving a login page and the possibility to select the right collection.

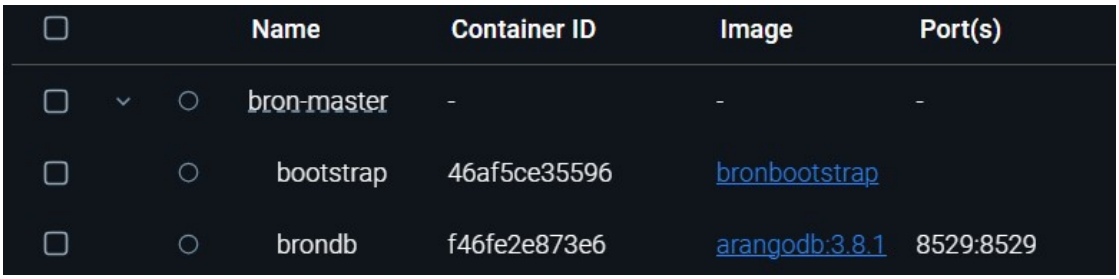

Figure 4.10. The containers running the BRON service.

The BRON database includes several collections, that are shown as in Figure 4.11 of documents, but we are interested to the CVE one, since it includes the metric *weight*, that refers to the CVSS score and can be found into the metadata, as shown in Figure 4.12. The CVSS score will be normalized and used to take the probability of the nodes of type vulExist as shown in Figure 4.13.

[3]https://github.com/ALFA-group/BRON

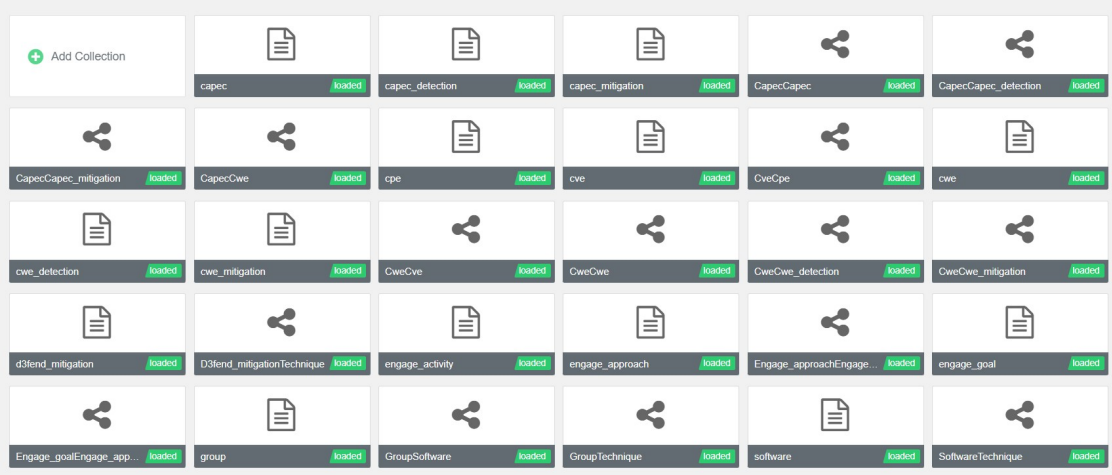

Figure 4.11. The BRON dashboard, presenting all the collections belonging to the database.

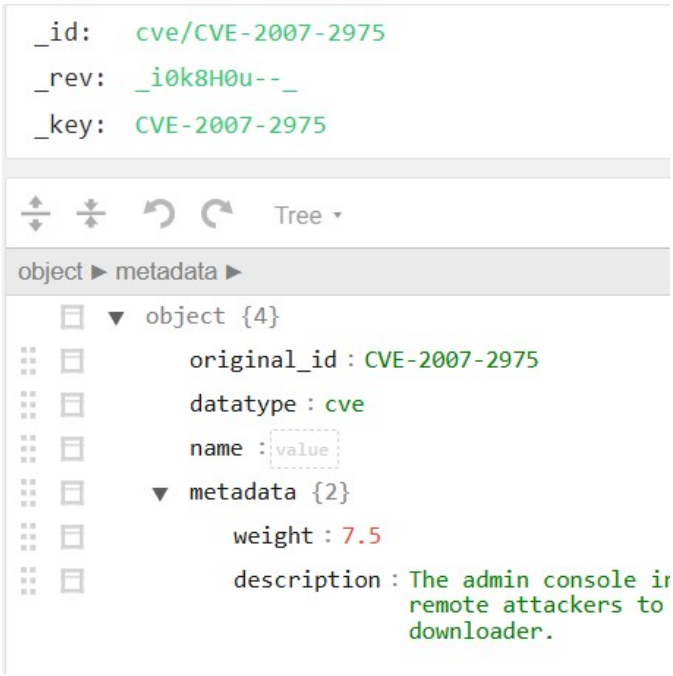

Figure 4.12. Each document gives information about a specific CVE, including the weight.

BRON is one of the two sources that is used to retrieve the probability of the LEAF nodes. However, it is not enough since we can also have LEAF nodes that are not strictly

```python
class CVSSCalculator:
    """Handle CVSS score calculations and conversions"""
    @staticmethod
    def normalize_cvss(cvss_score):
        """Convert CVSS score (0-10) to probability (0-1)"""
        return cvss_score / 10.0

    @staticmethod
    def get_cvss_probability(vuln_id):
        """Get probability from CVSS score"""

        score = get_cve_severity(vuln_id)
        return CVSSCalculator.normalize_cvss(score)
```

Figure 4.13. The CVSSCalculator class.

related to the existence of a vulnerability. To cope with this, the *running_rules.P* provides some probabilities by itself, attaching them to the rules. The MulValRuleExtractor shown in Figure 4.14 is able to read the predicate of the running rules using the method *_extract_rules* and to get the probability of the referring rule with the method *get_rule_probability* when a node references it, so that it is possible to have a marginal probability also for this kind of nodes.

```python
class MulValRuleExtractor:
    """Extract and manage probability rules from MulVAL"""
    def __init__(self, rules_file: str):
        self.rules = {}
        if rules_file and os.path.exists(rules_file):
            self._extract_rules(rules_file)

    def _extract_rules(self, rules_file):
        """Extract probability rules from MulVAL Prolog file"""
        with open(rules_file, 'r') as f:
            content = f.read()

        rule_pattern = r"interaction_rule\(\s*\((.*?)\),\s*rule_desc\('(.*?)',\s*([\d.]+)\)\)"
        matches = re.finditer(rule_pattern, content, re.DOTALL)

        for match in matches:
            rule_head = match.group(1).split(':-')[0].strip()
            description = match.group(2)
            probability = float(match.group(3))

            self.rules[rule_head] = {
                'description': description,
                'probability': probability
            }

    def get_rule_probability(self, predicate):
        """Get probability for a given predicate"""
        for rule_head, rule_info in self.rules.items():
            if rule_head.split('(')[0] in predicate:
                return rule_info['probability']
        return None

```

Figure 4.14. The MulValExtractor class.

Since the ABN needs each node has its CPD table to perform the inference, a default value is set in the case no matches are found. In fact, the code also assign a probability in the case of root nodes, namely nodes that haven't the characteristics of the above mentioned nodes. This kind of assignment is added to cope with potential errors of interpretation.

Figure 4.15, Figure 4.16 and Figure 4.17 show the code referring to the methods that actually build the CPD tables for the nodes.

Once the CPD tables are computed, the ABN is built and can be queried to perform the inference and to give back the marginal probabilities of each node, or the probability of the target node. The used method to perform the inference is the *VariableElimination*

```python
    def _create_and_cpd(self, node: int, node_info: dict, parents: list) -> TabularCPD:
        """Create CPD for AND nodes with rule-based probabilities"""
        n_parents = len(parents)
        cpt_values = []

        # Get rule probability if available
        rule_prob = self.rule_extractor.get_rule_probability(node_info['label'])
        success_prob = rule_prob if rule_prob is not None else 0.9

        for parent_combination in product([0, 1], repeat=n_parents):
            if all(parent_combination):
                prob_true = success_prob
            else:
                prob_true = 0.1
            cpt_values.append([1 - prob_true, prob_true])

        return TabularCPD(
            variable=node,
            variable_card=2,
            values=np.array(cpt_values).T.tolist(),
            evidence=parents,
            evidence_card=[2] * n_parents
        )
```

Figure 4.15. The method to create the CPD for AND nodes.

one, provided by PgmPy, for the reason explained in Section 2.5.

```python
    def _create_or_cpd(self, node: int, node_info: dict, parents: list) -> TabularCPD:
        """Create CPD for OR nodes with rule-based probabilities"""
        n_parents = len(parents)
        cpt_values = []

        # Get rule probability if available
        rule_prob = self.rule_extractor.get_rule_probability(node_info['label'])
        max_prob = rule_prob if rule_prob is not None else 0.9

        for parent_combination in product([0, 1], repeat=n_parents):
            true_parents = sum(parent_combination)
            if true_parents > 0:
                # Scale probability based on number of true parents
                prob_true = min(max_prob, 0.3 + ((max_prob - 0.3) * true_parents / n_parents))
            else:
                prob_true = 0.1
            cpt_values.append([1 - prob_true, prob_true])

        return TabularCPD(
            variable=node,
            variable_card=2,
            values=np.array(cpt_values).T.tolist(),
            evidence=parents,
            evidence_card=[2] * n_parents
        )
```

Figure 4.16. The method to create the CPD for OR nodes.

```python
    def _create_leaf_cpd(self, node: int, node_info: dict) -> TabularCPD:
        """Create CPD for leaf nodes with enhanced probability calculation"""
        base_prob = node_info['value']

        if 'vulExists' in node_info['label']:
            base_prob = self._get_vulnerability_probability(node_info['label'])
        elif node_info['value'] == 1:  # Explicit LEAF node
            base_prob = 1.0
        else:
            rule_prob = self.rule_extractor.get_rule_probability(node_info['label'])
            if rule_prob is not None:
                base_prob = rule_prob

        return TabularCPD(
            variable=node,
            variable_card=2,
            values=[[1 - base_prob], [base_prob]]
        )
```

Figure 4.17. The method to create the CPD for LEAF nodes.

Chapter 5

Experimental Results

5.1 Proof-of-Concept of the ABN Generator

This chapter provides an example of how the proposed tool works and how the results are presented. As it is a PoC, the following case is a toy example. Moreover, the probabilities are computed as explained in the previous chapter, but the strength of the tool is that it can be potentially enriched with many other metrics and enterprises provided data, so that, even if they are meaningless in this thesis, it may be further improved to become meaningful instead.

The following code is the content of the input.P file, provided by the authors for testing.

```
1  attackerLocated(internet).
2  attackGoal(execCode(workStation,_)).
3
4  hacl(internet, webServer, tcp, 80).
5  hacl(webServer, _,  _, _).
6  hacl(fileServer, _, _, _).
7  hacl(workStation, _, _, _).
8  hacl(H,H,_,_).
9
10 /* configuration information of fileServer */
11 networkServiceInfo(fileServer, mountd, rpc, 100005, root).
12 nfsExportInfo(fileServer, '/export', _anyAccess, workStation).
13 nfsExportInfo(fileServer, '/export', _anyAccess, webServer).
14 vulExists(fileServer, vulID, mountd).
15 vulProperty(vulID, remoteExploit, privEscalation).
16 localFileProtection(fileServer, root, _, _).
17
18 /* configuration information of webServer */
19 vulExists(webServer, 'CAN-2002-0392', httpd).
20 vulProperty('CAN-2002-0392', remoteExploit, privEscalation).
21 networkServiceInfo(webServer , httpd, tcp , 80 , apache).
22
23 /* configuration information of workStation */
24 nfsMounted(workStation, '/usr/local/share', fileServer, '/export
      ', read).
```

input.P describes the 3 host scenario that is tested in their paper and shown in Figure 5.1. The senario represents a network architecture divided into several zones, each with specific components and security configurations:

- The **Internet**, depicted as the entry point for a potential attacker (`attackerLocated(internet)`).
- The **Demilitarized Zone (DMZ)**, located between two firewalls:
  - **fw1** controls traffic from the Internet to the DMZ.
  - **fw2** regulates traffic from the DMZ to the Internal Network.

  The DMZ includes a **Web Server** exposed to the Internet but isolated from the Internal Network. It runs an HTTP service on port 80 (`networkServiceInfo(webServer,httpd,tcp,80,apache)`) and is vulnerable to a known exploit (`vulExists(webServer, 'CAN-2002-0392', httpd)`) that allows remote privilege escalation (`vulProperty('CAN-2002-0392',remoteExploit,privEscalation)`).
- The **Internal Network** contains critical assets:
  - The **Workstation** used by employees has an NFS mount to the file server for shared file access (`nfsMounted(workStation,'/usr/local/share',fileServer,'/export',read)`).
  - The **File Server** provides storage services via an NFS service on port 100005 (`networkServiceInfo(fileServer,mountd,rpc,100005,root)`). It is vulnerable to an unidentified remote exploit (`vulExists(fileServer,vulID,mountd)`) enabling privilege escalation (`vulProperty(vulID,remoteExploit,privEscalation)`). The server exports the **/export** directory to both the Workstation and Web Server (`nfsExportInfo(fileServer,'/export',_anyAccess,workStation)` and `nfsExportInfo(fileServer,'/export',_anyAccess,webServer)`).
  - The **Project Plan**, **Web Pages**, and **Binaries** are internal resources critical to the organization.

The attacker's goal is to execute code on the workStation (`attackGoal(execCode(workStation,_))`). The scenario represents the attacker aiming to compromise an employee's workstation to gain access to sensitive data or further infiltrate the network. Furthermore, the hacl (host access control list) predicates define the network access rules. In particular the Internet can access the webServer on port 80 (`hacl(internet,webServer,tcp,80)`), the webServer, fileServer, and workStation have unrestricted access to themselves (`hacl(H,H,_,_)`). The fileServer and workStation have additional access rules, allowing NFS communication and file sharing.

Using the MulVAL framework, the scenario can be analyzed to identify potential attack paths. MulVAL will generate the attack graph in Figure 5.2, also producing the equivalent csv files. The latter will be used by the tool to generate the ABN. From the figure, it is

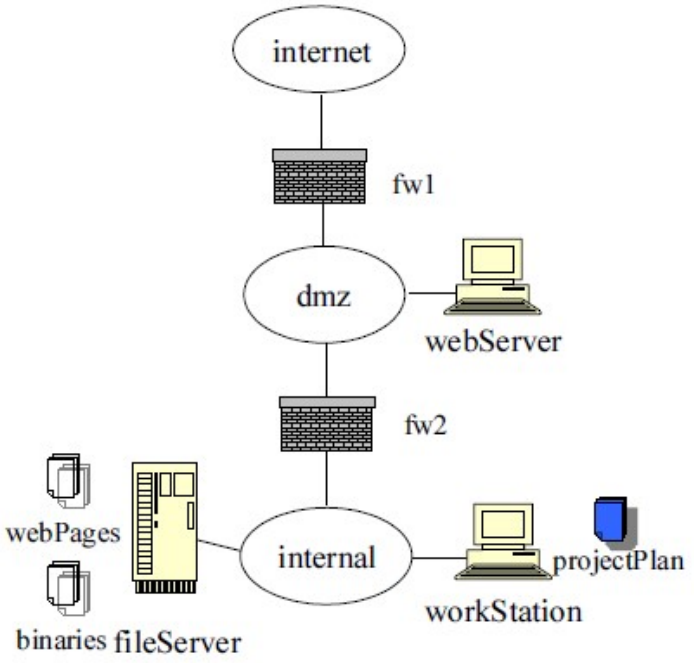

Figure 5.1. Multi host scenario.

possible to visualize three types on nodes: the rectangular ones are the LEAF nodes, the rhomboidal ones the OR nodes, while the circular ones the AND nodes.

The tool is able to automatically compute the CPD tables and to print out the results in the form of tables. In the test case, node 1 (`execCode(workstation,root)`) is indeed the target node, representing the attacker's ultimate goal: executing code with root privileges on the workstation.

The attack path can be summarized in this way:

attackerLocated(internet) (18)
↓
hacl(internet,webServer,tcp,80) (17)
↓
RULE 6 (direct network access) (16)
↓
netAccess(webServer,tcp,80) (15)
↓
RULE 2 (remote exploit) (14) → vulExists(webServer,'CAN-2002-0392',httpd) (20)
↓
execCode(webServer,apache) (13)
↓
RULE 5 (multi-hop access) (11) → hacl(webServer,fileServer,rpc,100005) (12)

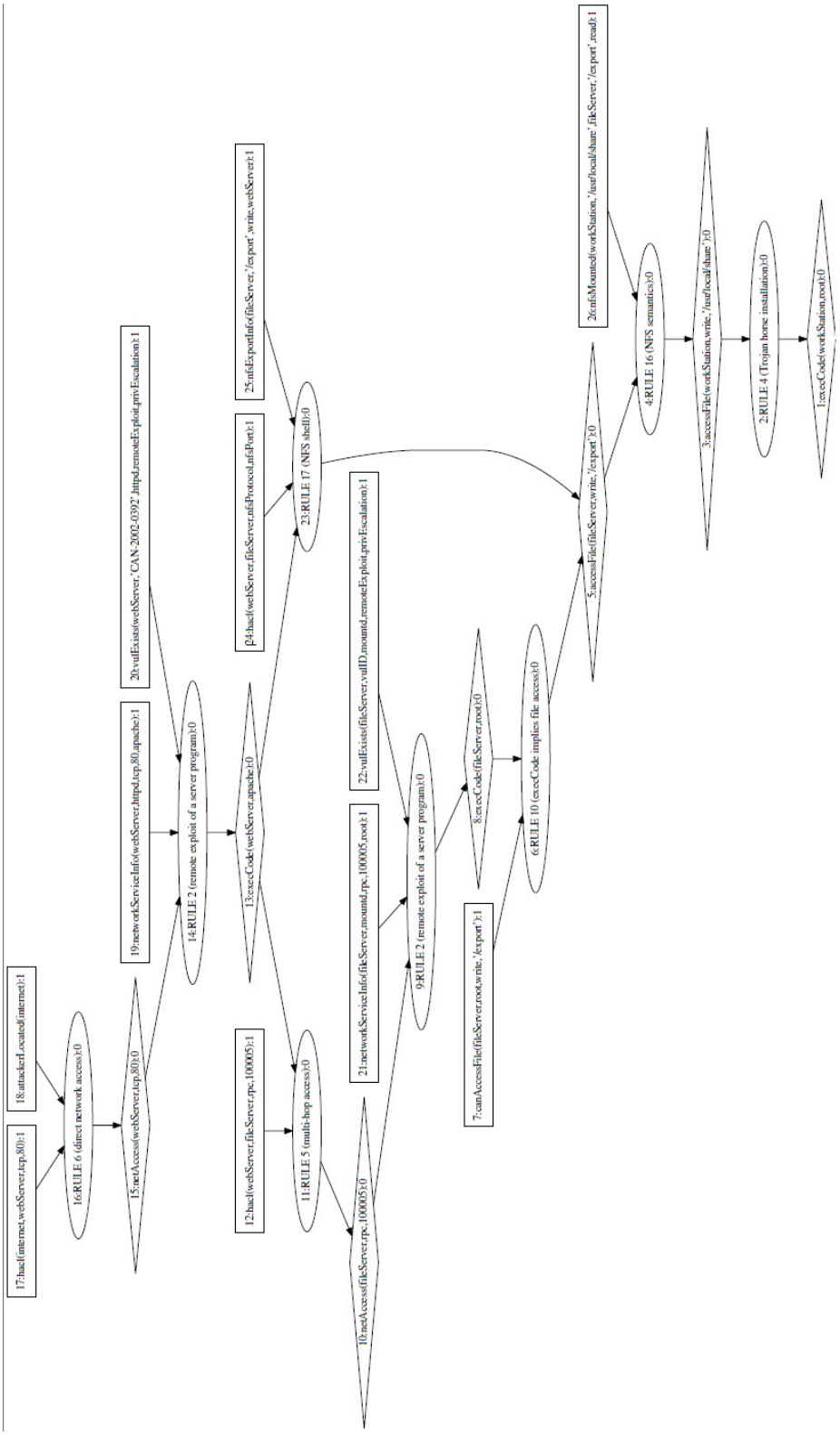

Figure 5.2. The MulVAL's output, represented with GraphViz.

```
+-------+-----------+

Node 13 (execCode(webServer,apache)):
+-------+-----------+
| 13    |   phi(13) |
+=======+===========+
| 13(0) |    0.6416 |
+-------+-----------+
| 13(1) |    0.3584 |
+-------+-----------+

Node 14 (RULE 2 (remote exploit of a server program)):
+-------+-----------+
| 14    |   phi(14) |
+=======+===========+
| 14(0) |    0.3540 |
+-------+-----------+
| 14(1) |    0.6460 |
+-------+-----------+

Node 15 (netAccess(webServer,tcp,80)):
+-------+-----------+
| 15    |   phi(15) |
+=======+===========+
| 15(0) |    0.0900 |
+-------+-----------+
| 15(1) |    0.9100 |
+-------+-----------+

Node 16 (RULE 6 (direct network access)):
+-------+-----------+
| 16    |   phi(16) |
+=======+===========+
| 16(0) |    0.1000 |
+-------+-----------+
| 16(1) |    0.9000 |
+-------+-----------+
```

Figure 5.3. The marginal probability output.

↓
netAccess(fileServer,rpc,100005) (10)
↓
RULE 2 (remote exploit) (9) → vulExists(fileServer,vulID,mountd) (22)
↓
execCode(fileServer,root) (8)
↓
RULE 10 (execCode implies file access) (6) → accessFile(fileServer,write,'/export') (5)
↓
RULE 17 (NFS shell) (23) → nfsExportInfo(fileServer,'/export',write,webServer) (25)

↓
accessFile(workStation,write,'/usr/local/share') (3)
↓
RULE 4 (Trojan horse installation) (2)
↓
execCode(workStation,root) (1)

```
Probability analysis for target node 1:
+------+----------+
| 1    |   phi(1) |
+======+==========+
| 1(0) |   0.7025 |
+------+----------+
| 1(1) |   0.2975 |
+------+----------+
```

Figure 5.4. The target node probability.

Finally, the marginal probability of each node, as in Figure 5.3, followed by the target node's probability, Figure 5.4 are printed out. We can observe that, with the analyzed test case, at the end the probability to perform the arbitrary code execution on the workstation has a probability of 29%.

Chapter 6

Conclusion

This Thesis proposes a framework for cybersecurity risk assessment, offering a foundational pipeline that integrates ontologies, MulVAL, and Bayesian Networks to compute the probabilities of attack patterns. The approach leverages formal ontologies to model relationships between assets and vulnerabilities, MulVAL for automated attack graph generation, and Bayesian Networks to dynamically assess the likelihood of attack success. It opens the door to further improvements, even including the exploration of Artificial Intelligence (AI) integration within this pipeline, highlighting its potential to enhance threat prediction, automate probability calibration, and enable adaptive risk modeling in future studies.

While the framework is implemented as a proof of concept (PoC), it faces limitations inherent to early-stage research. A significant challenge lies in the assignment of node probabilities, which currently relies on manual input or simplified heuristics. To address this, the thesis identifies opportunities to integrate established frameworks such as MITRE's CAPEC (for attack pattern taxonomies) and NVD's CVSS (for standardized vulnerability scoring). Synergies between these resources could enable automated probability derivation, ensuring consistency and scalability in real-world deployments.

Furthermore, the proposed model does not yet incorporate security controls (e.g., firewalls, intrusion detection systems) that could mitigate attack success probabilities. Future work should explore dynamic adjustments to Bayesian Network parameters based on implemented controls, enabling "what-if" analyses for defence optimization. This aligns with emerging research in adaptive cybersecurity frameworks, where AI-driven systems continuously update risk assessments in response to evolving threats and countermeasures.

The pipeline's modular design allows for incremental improvements, such as:

- **AI-driven threat intelligence**, using machine learning to correlate historical attack data with real-time network telemetry.
- **Automated CPD generation**, leveraging CVSS scores or attack simulation tools (e.g., Metasploit) to infer conditional probabilities.
- **Integration with SIEM systems**, hence embedding the framework into security operations centers for proactive risk prioritization.

While challenges remain, particularly in scalability and validation across heterogeneous environments, this work establishes a methodological foundation for unifying semantic modeling, attack graph analysis, and probabilistic reasoning in cybersecurity risk assessment.

Bibliography

[1] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. «Scalable, graph-based network vulnerability analysis». In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. 2002, pp. 217–224.

[2] Ankur Ankan and Abinash Panda. *pgmpy: Probabilistic Graphical Models using Python*. https://pgmpy.org/. Accessed: 2025-03-09. 2015.

[3] ArangoDB GmbH. *ArangoDB: A multi-model database*. https://www.arangodb.com/. Accessed: 2025-03-09.

[4] Jennifer Cawthra, Michael Ekstrom, Lauren Lusty, Julian Sexton, John Sweetnam, and Anne Townsend. «NIST SPECIAL PUBLICATION 1800-26A-Data Integrity:-Detecting and Responding to Ransomware and Other». In: (2020).

[5] S. Ceri, G. Gottlob, and L. Tanca. «What you always wanted to know about Datalog (and never dared to ask)». In: *IEEE Transactions on Knowledge and Data Engineering* 1.1 (1989), pp. 146–166. DOI: 10.1109/69.43410.

[6] World Wide Web Consortium et al. «OWL 2 web ontology language document overview». In: (2012).

[7] Nicola d'Ambrosio, Gaetano Perrone, and Simon Pietro Romano. «Including insider threats into risk management through Bayesian threat graph networks». In: *Computers & Security* 133 (2023), p. 103410.

[8] Fabio De Rosa, Nicoló Maunero, Luca Nicoletti, Paolo Prinetto, Martina Trussoni, et al. «Ontology for Cybersecyrity Governance of ICT Systems». In: *ITASEC*. 2022, pp. 52–63.

[9] Fabio De Rosa, Nicolò Maunero, Paolo Prinetto, Federico Talentino, and Martina Trussoni. «ThreMA: Ontology-Based Automated Threat Modeling for ICT Infrastructures». In: *IEEE Access* 10 (2022), pp. 116514–116526. DOI: 10.1109/ACCESS.2022.3219063.

[10] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. «Graphviz—open source graph drawing tools». In: *Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers 9*. Springer. 2002, pp. 483–484.

[11] Stefan Fenz. «An ontology-and bayesian-based approach for determining threat probabilities». In: *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*. 2011, pp. 344–354.

[12] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. «Exploring Network Structure, Dynamics, and Function Using NetworkX». In: *Proceedings of the 7th Python in Science Conference (SciPy2008)* (2008), pp. 11–15. URL: https://networkx.github.io/documentation/stable/.

[13] Chris Hankin, Pasquale Malacaria, et al. «Attack dynamics: An automatic attack graph generation framework based on system topology, CAPEC, CWE, and CVE databases». In: *Computers & Security* 123 (2022), p. 102938.

[14] Ian Horrocks, Peter F Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean, et al. «SWRL: A semantic web rule language combining OWL and RuleML». In: *W3C Member submission* 21.79 (2004), pp. 1–31.

[15] Sushil Jajodia, Steven Noel, and Brian O'berry. «Topological analysis of network attack vulnerability». In: *Managing Cyber Threats: Issues, Approaches, and Challenges* (2005), pp. 247–266.

[16] Christopher S. Johnson, Mark Lee Badger, David A. Waltermire, Julie Snyder, and Clem Skorupka. *Guide to Cyber Threat Information Sharing:* en. 2016. DOI: https://doi.org/10.6028/NIST.SP.800-150.

[17] Pengfei Li and Xiaofeng Qiu. «NodeRank: An algorithm to assess state enumeration attack graphs». In: *2012 8th International Conference on Wireless Communications, Networking and Mobile Computing*. IEEE. 2012, pp. 1–5.

[18] Tong Li, Elda Paja, John Mylopoulos, Jennifer Horkoff, and Kristian Beckers. «Security attack analysis using attack patterns». In: *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*. 2016, pp. 1–13. DOI: 10.1109/RCIS.2016.7549303.

[19] Nicolò Maunero, Fabio De Rosa, and Paolo Prinetto. «Towards Cybersecurity Risk Assessment Automation: an Ontological Approach». In: *2023 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. 2023, pp. 0628–0635. DOI: 10.1109/DASC/PiCom/CBDCom/Cy59711.2023.10361456.

[20] Wes McKinney. «Data Structures for Statistical Computing in Python». In: *Proceedings of the 9th Python in Science Conference (SciPy2010)* (2010), pp. 56–61. URL: https://pandas.pydata.org/.

[21] Mark A Musen. «The protégé project: a look back and a look forward». In: *AI matters* 1.4 (2015), pp. 4–12.

[22] Michael Nieles, Kelley Dempsey, Victoria Yan Pillitteri, et al. «An introduction to information security». In: *NIST special publication* 800.12 (2017), p. 101.

[23] Joohwan Oh and ArangoDB Community. *ArangoDB-Python: A Python Driver for ArangoDB*. https://github.com/joowani/arango. Accessed: 2025-03-09. 2025.

[24] Xinming Ou, Sudhakar Govindavajhala, Andrew W Appel, et al. «MulVAL: A logic-based network security analyzer.» In: *USENIX security symposium*. Vol. 8. Baltimore, MD. 2005, pp. 113–128.

[25] Celia Paulsen. *Glossary of key information security terms*. Tech. rep. National Institute of Standards and Technology, 2018.

[26] R.W. Ritchey and P. Ammann. «Using model checking to analyze network vulnerabilities». In: *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*. 2000, pp. 156–165. DOI: 10.1109/SECPRI.2000.848453.

[27] Ronald Ross. *Guide for Conducting Risk Assessments*. en. 2012. DOI: https://doi.org/10.6028/NIST.SP.800-30r1.

[28] Konstantinos Sagonas, Terrance Swift, and David S Warren. «Xsb: An overview of its use and implementation». In: *SUNY Stony Brook* (1993), p. 11794.

[29] Jonathan Spring, Eric Hatleback, A Manion, and D Shic. «Towards improving CVSS». In: *SEI, CMU, Tech. Rep* (2018).

[30] Todd Andrew Stephenson. «An introduction to Bayesian network theory and usage». In: (2000).

[31] L.P. Swiler, C. Phillips, D. Ellis, and S. Chakerian. «Computer-attack graph generation tool». In: *Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01*. Vol. 2. 2001, 307–321 vol.2. DOI: 10.1109/DISCEX.2001.932182.

[32] Guido Van Rossum and Fred L. Jr. Drake. *Python Language Reference, version 3.9*. https://docs.python.org/3/library/tkinter.html. Accessed: 2025-03-09. 2020.

[33] Kengo Zenitani. «Attack graph analysis: An explanatory guide». In: *Computers & Security* 126 (2023), p. 103081.