# POLITECNICO DI TORINO

**MASTER's Degree in Computer Engineering**


**MASTER's Degree Thesis**

# A Hybrid Model for Threat Analysis Enhanced by LLM: Integrating MITRE ATT&CK for Cyber-Physical System Security


| Supervisors | Candidate |
| --- | --- |
| Prof. Fulvio VALENZA | Vittorio SANFILIPPO |
| Prof. Erisa KARAFILI | |


**April 2025**

# Abstract

In an era in which smart systems intrinsically integrate cyber, physical, and human components, threat analysis requires an approach that overcomes the limitations of traditional models—not by considering purely cyber threats but by focusing on the interaction between these domains. Based on the hybrid model described in "A Hybrid Threat Model for Smart Systems" and on the TAMELESS tool—an automatic tool that, through formal rules, derives the security state of complex systems—this thesis aims to enhance TAMELESS to make it more exhaustive, efficient, and accessible.

Initially, the study focused on analyzing real cases involving interactions between cyber, physical, and human elements, verifying the applicability of TAMELESS rules to these scenarios. Subsequently, an analysis was conducted on the nature of the properties and relationships, which highlighted the coherence of the relationships defined in the model. This analysis extended to the formal verification of the rules using the PRISM tool, a model checker for the formal modeling and analysis of systems that exhibit random or probabilistic behavior, used to verify the activation, correctness, and consistency of the derivation system.

In parallel, a modification was made to the TAMELESS system by implementing a graphical interface that allows for an intuitive visualization of the graph representing the system to be analyzed and the resulting attack graph. An update was made to the original TAMELESS code, enabling communication with the new versions of the Neo4J graph database. Furthermore, during the study it was decided to make the tool more exhaustive in interpreting the results; for this reason, a change was made to the vulnerability analysis workflow, integrating a module based on LLAMA 3.1 with RAG technology that, leveraging the MITRE ATT&CK dataset, extracts and subsequently verifies the applicability of attack techniques to the

individual nodes of the system. The analysis performed in parallel by the Large Language Model allows for the merging of the results derived from TAMELESS with the information coming from MITRE ATT&CK, not only partially reducing the graph and facilitating the identification of attack paths, but also suggesting specific detection and mitigation methods and calculating the probability and risk associated with each node.

The use of LLMs, increasingly impactful nowadays, allows for a significant improvement in understanding the results and compromise processes in hybrid smart systems, without distorting the formal nature of the system. Moreover, it makes TAMELESS accessible to a wider audience of users, alongside well-known and established methodologies such as MITRE ATT&CK.

# Acknowledgements

*I gratefully acknowledge the use of the IRIDIS High Performance Computing Facility, and the excellent support services at the University of Southampton, in completing this work. In addition, I wish to thank Professor Karafili, who guided me step by step through weekly meetings as I worked on my thesis and was a fundamental point of reference during my time abroad. I am also grateful to Professor Valenza, who believed in me and made this opportunity possible.*

*Desidero ringraziare tutta la mia famiglia, pilastro essenziale della mia vita, che considero il nido che mi ha cresciuto e il rifugio in cui torno sempre. In particolare, desidero ringraziare le nonne, Antonella e Grazia, che mi hanno accudito e amato nei momenti più importanti della mia vita.*

*Ringrazio i miei genitori, a cui devo tutto e che, in un momento difficile come quello affrontato negli ultimi anni, hanno sempre trovato la forza e l'energia per guidarmi, sostenermi e aiutarmi a raggiungere i miei obiettivi.*

*Ringrazio inoltre Marta e Massimo che, pur avendo ruoli diversi, non considero più semplicemente 'amici', ma letteralmente parte della mia famiglia e del mio mondo. Grazie per le interminabili chiamate, sempre pronti a regalarmi una risata, ad ascoltarmi o darmi un semplice consiglio, specialmente durante questo periodo all'estero.*

*Infine, ringrazio chi da lassù mi guarda e mi protegge ogni giorno. So che sarebbero molto orgogliosi dei traguardi raggiunti.*

# Table of Contents

# List of Tables

# List of Figures

# Acronyms

**IoT**

Internet of Things

**CPS**

Cyber Physical System

**ICS**

Industrial Control System

**GUI**

Graphical User Interface

**TAMELESS**

Threat & Attack ModEL Smart System

**LLM**

Large Language Model

**LLaMA**

Large Language Model Meta AI

**RAG**

Retrieval-Augmented Generation

**DTMC**

Discrete-Time Markov Chain

**FAISS**

Facebook AI Similarity Search

**DAO**

Data Access Object

# Chapter 1

# Introduction

## 1.1 Context and Objectives

Nowadays, within a context of continuous constant technological development and a new industrial revolution underway, is possible to see a growing expansion of elements that need an intrinsic combination of human, physical and cyber elements. A vivid example are devices called the Internet of Things (IoT), which are identified as a "network of physical objects that are connected to the internet using software, sensors, and other technologies". This enables these "smart" systems to collect and share data, making communication and automation possible. These technologies are not only applicable for 'smart houses' or wearable devices (smart watches), but they are broadly used in the industrial realities too. Within the pool of IoT landscape there exist systems known as "Cyber Physical Systems" (CPS), that are, engineered systems that integrate computational elements with physical processes, creating a tight interconnection between the digital and physical worlds. Referring specifically to the industrial domain are defined "Industrial Control Systems" (ICS), systems that incorporate hardware, software, and network connectivity to observe and begin to take control of production processes within manufacturing, critical infrastructure, and other industries settings.
However, the integration of such systems requires a broader reflection on the possible threats affecting these new devices. Attacks today are not only limited to components in the cyber domain, but target physical vulnerabilities and aspects of

human behavior. Traditional threat analysis methodologies, which are predominantly focused on the cyber aspect, fail to capture the interaction and propagation of threats across these domains, not considering alternative paths for the tampering of non-cyber components, and thus leaving fundamental aspects of the overall system security uncovered. For example, in an industrial plant, unauthorized physical access can facilitate an attack on a digital control system, while a human error, such as a phishing campaign, can compromise sensitive data and pave the way for further attacks. For this reason, it is necessary to adopt a holistic and integrated approach, capable of modeling the interdependencies between components and supporting the analysis of scenarios in which hybrid attacks, combining physical, digital, and human elements, can be anticipated and effectively countered.

Given the context that has been outlined, the thesis work is based on the hybrid threat model described in "A Hybrid Threat Model for Smart Systems" [1] and on the use of the TAMELESS tool, which leverages formal rules to derive the security state of complex systems. The main objective of the work is to enhance TAMELESS, making it more comprehensive and accessible through the integration of new technologies and methodologies aimed at achieving a deeper threat analysis, defining metrics such as risk and probability for the various identified threats, without losing the formal component of the model. In particular, the study began by addressing the analysis of smart systems, looking for and examining real-world cases in which cyber, physical, and human components interact, and demonstrating how TAMELESS can identify the critical elements in these examples, thereby proving that the approach it employs is effective even for complex systems. Another aspect of the work involves further enhancing TAMELESS.
First, given the current lack of formal verification of the model rules, a formal verification process is carried out using the PRISM tool, which validates the derivation rules and ensures the accuracy and soundness of the model. In addition, an intuitive GUI has been implemented for the visualization of attack graphs, simplifying both the input of data representing the system and the interpretation of the results by the user. The tool has also been updated to integrate the latest versions of the Neo4J graph database, which improves the management of relationships and ensures compatibility with new system versions.
Finally, the thesis introduces a new vulnerability analysis module based on a

Large Language Model, which utilizes LLaMA 3.1 with RAG technology to extract techniques from the MITRE ATT&CK dataset, verifying and integrating the attack techniques applicable to individual entities. This analysis, conducted in parallel with the analysis carried out by TAMELESS using a LLM, not only makes it possible to partially reduce the attack graph by improving the visualization of attack paths, but also suggests specific methods of detection and mitigation by calculating the probability and risk associated with each technique.
The adoption of LLM models and the integration with established methodologies such as MITRE ATT&CK make TAMELESS a tool accessible to a wider audience, expanding the possibilities for application in different contexts and contributing to a more exhaustive approach to the security of complex systems.

## 1.2 Structure of the Thesis

The body of the thesis is organized as follows:

- **Chapter 2 – Background Context:** In this chapter, an overview of currently available threat models is provided, explaining the need for the use of the model presented on 'A Hybrid Threat Model for Smart Systems'[1]. An overall view of the TAMELESS tool is then provided and its functionality, concluding with a presentation of the other tools used in this work.

- **Chapter 3 – Case Studies and Evaluation of the Model:** This chapter describes the examples identified that will later be tested by the new system.

- **Chapter 4 – Formal Verification of the Model:** This chapter presents an overview of the limitations already present in the model and its rules and relationships. In addition, the study conducted on the nature of the relationships and interactions with the various domains is shown, and finally the formal verification with the PRISM tool is presented, defining the model used and discussing the results obtained.

- **Chapter 5 – Technological Integration of TAMELESS:** This chapter illustrates the main modifications made to TAMELESS, showing the changes applied to the original code and presenting the new graphical interface.

- **Chapter 6 – Enhanced TAMELESS:** This chapter presents the main contribution of this thesis, defining the new threat analysis workflow with the addition of the LLaMA-based vulnerability scanning module. It also shows how the probability and risk metrics defined for each node and technique identified are determined.
- **Chapter 7 – Validation:** This chapter illustrates the results obtained from the new workflow, validating them alongside the new metrics acquired, and reflecting on the contribution achieved by the introduction of the new module.
- **Chapter 8 – Conclusions and Future Works:** This final chapter summarizes the results of the thesis, also presenting the limitations of the work, and proposes directions for future research.

# Chapter 2

# Background Context

## 2.1 Threat Analysis Models and Hybrid Threat Model

Threat modeling is a term applied to techniques used for modeling and analyzing IT systems with the purpose of identifying how the system or its affiliated services can be attacked. It is mainly implemented during the design and development phases of a system's or technological service's lifecycle, allowing for the necessary modifications to enhance the system's security as much as possible.

Nowadays, there are several types of threat models, each with different characteristics, that can be used together to make the analysis more reliable. The paper "Threat Modeling: A Summary of Available Methods." [2] presents a fairly comprehensive overview of the main threat modeling methods currently available on the market, ranging from the more traditional and well-known approaches, such as STRIDE to some methods that can be considered hybrid, depending on how they are used, such as OCTAVE.

Below, still taken from the paper, a summary table identifying the main features for each model is extracted.

**Table 2.1:** Threat Modeling Methods Features. Adapted from [2]

| Threat Modeling Method | Features |
|---|---|
| **STRIDE** | • Helps identify relevant mitigating techniques.<br>• Is the most mature.<br>• Is easy to use but is time consuming. |
| **PASTA** | • Helps identify relevant mitigating techniques.<br>• Directly contributes to risk management.<br>• Encourages collaboration among stakeholders.<br>• Contains built-in prioritization of threat mitigation.<br>• Is laborious but has rich documentation. |
| **LINDDUN** | • Helps identify relevant mitigating techniques.<br>• Contains built-in prioritization of threat mitigation.<br>• Can be labor intensive and time consuming. |
| **CVSS** | • Contains built-in prioritization of threat mitigation.<br>• Has consistent results when repeated.<br>• Automated components.<br>• Has score calculations that are not transparent. |
| **Attack Trees** | • Helps identify relevant mitigating techniques.<br>• Has consistent results when repeated.<br>• Is easy to use if you already have a thorough understanding of the system. |

| Threat Modeling Method | Features |
|---|---|
| **Persona non Grata** | • Helps identify relevant mitigating techniques.<br>• Directly contributes to risk management.<br>• Has consistent results when repeated.<br>• Tends to detect only some subsets of threats. |
| **Security Cards** | • Encourages collaboration among stakeholders.<br>• Targets out-of-the-ordinary threats.<br>• Leads to many false positives. |
| **hTMM** | • Contains built-in prioritization of threat mitigation.<br>• Encourages collaboration among stakeholders.<br>• Has consistent results when repeated. |
| **Quantitative TMM** | • Contains built-in prioritization of threat mitigation.<br>• Has automated components.<br>• Has consistent results when repeated. |
| **Trike** | • Helps identify relevant mitigating techniques.<br>• Directly contributes to risk management.<br>• Contains built-in prioritization of threat mitigation.<br>• Encourages collaboration among stakeholders.<br>• Has automated components.<br>• Has vague, insufficient documentation. |

| Threat Modeling Method | Features |
|---|---|
| **VAST Modeling** | • Helps identify relevant mitigating techniques.<br>• Directly contributes to risk management.<br>• Contains built-in prioritization of threat mitigation.<br>• Encourages collaboration among stakeholders.<br>• Has consistent results when repeated.<br>• Has automated components.<br>• Is explicitly designed to be scalable.<br>• Has little publicly available documentation. |
| **OCTAVE** | • Helps identify relevant mitigating techniques.<br>• Directly contributes to risk management.<br>• Contains built-in prioritization of threat mitigation.<br>• Encourages collaboration among stakeholders.<br>• Has consistent results when repeated.<br>• Is explicitly designed to be scalable.<br>• Is time consuming and has vague documentation. |

The threat modeling methods listed above are very effective for traditional IT systems, but they fail to comprehensively capture the threats that affect smart systems. In particular, hTMM is the only one among the proposed models labeled as the "Hybrid Threat Modeling Method," yet it does not sufficiently integrate, like the other models, the interactions among the human, cyber, and physical domains typical of smart systems; therefore, it is not applicable to our scenarios. In fact, a hybrid threat model is defined as a system that integrates traditional and modern threat modeling techniques through the combined use of traditional risk assessment and innovative data-driven analysis, allowing simultaneous consideration of the IT, physical, and human dimensions and providing a complete overview of the vulnerabilities and threats that can affect the system.

## 2.2 TAMELESS

Given the issues presented in the previous paragraph, is provided an overview of TAMELESS (Threat & Attack ModEL Smart System)[3], an automated tool designed specifically to address these challenges in smart systems which relies on an XSB Prolog interpreter[4]. TAMELESS implements a hybrid threat model that explicitly incorporates human, cyber, and physical components, along with the relationships between them, into its analysis.

The model allows the representation of entities and threats of different nature in a single graph representing the system to be analyzed,specifying the security properties of these entities, that "represent how the security state of the components can change" and the relationships between the components[1]. For entities and threats, the authors define: "An entity is a system or system component that can be of a cyber, physical, or human nature" and "A threat is one or a sequence of actions that directly or indirectly changes a property that can alter the security state of an entity"[1]. Using a set of defined derivation rules, TAMELESS automatically infers additional, derived properties.

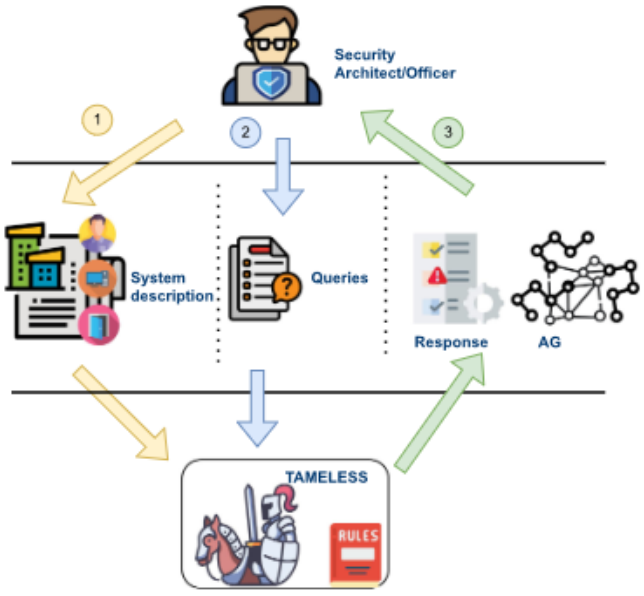


**Figure 2.1:** TAMELESS's architecture and workflow (Adapted from [1]).

Figure 2.1 illustrates the structure of TAMELESS: the security architect first inputs the nodes representing the system, then requests the start of the analysis by issuing a query to the system. At this point, Prolog will verify that the rules defined in the threat model can be applied, thereby determining the various derived properties for the nodes in the graph. After that, the user will be able to see the results of the analysis and verify the state of the various entities of the system.

The analysis performed with TAMELESS is static, meaning it is based on fixed information about the system; if these were to change (for example, the insertion of a new node), the analysis must be repeated. This must also be done if the user wants to change any information in the system, to verify that any countermeasures are effective.

In conclusion, TAMELESS allows for effectively verifying the security state of the various components of a smart system, taking into account the different natures of the elements. In the following chapters, an in-depth analysis of the properties, relationships, and rules defined in the model will be addressed.

## 2.3 Support Technologies and Tools

Now, the tools that were used for the project's application part are introduced, providing an overview of their usage.

### 2.3.1 MITRE ATT&CK Framework

**MITRE ATT&CK** is a framework developed by MITRE with the goal of collecting the techniques and tactics used by attackers based on real-world observations [5]. To facilitate the identification of various threats, the framework is organized hierarchically. It primarily defines three macro areas, known as technology domains:

- **Enterprise:** representing traditional enterprise networks and cloud technologies;
- **Mobile:** for mobile communication devices;
- **ICS:** for industrial control systems.

Thanks to these domains, there is a clear distinction between the different types of threats that can be exploited. Each domain contains elements called **tactics**, which represent the "why" or the reason an adversary is performing an action;

examples include Initial Access, Lateral Movement, Defense Evasion, etc.

Each tactic, in turn, includes the **techniques** that represent "how" adversaries achieve tactical goals by performing an action. This is the core of the information that provides a detailed description of the threat, enabling the user to thoroughly understand its functioning and how it might be exploited. Each technique is identified by a specific ID in the format **Txxxx** (for example, T0827 Loss of Control). In addition to the threat description, a technique also carries other useful information for understanding the attack, such as the platforms, indicating which systems are vulnerable to that specific threat. Moreover, there is the presence of **sub-techniques**, which is a more specific or lower-level description of adversarial behavior, and finally, real-life examples of where that threat has been exploited. One of the most useful pieces of information, which will be utilized in our work, concerns **mitigation** and **detection**. Indeed, each technique or sub-technique can be linked to these aspects, thus providing important details to best defend the system.

Finally, to make all this information easily accessible, the ATT&CK framework provides what are called "Matrix", a tabular representation where the columns corresponde to the tactics and each row lists the various techniques. In this case, a matrix will not be used to extract the data; instead, the JSON files directly accessible from the ATT&CK Navigator GitHub repository [6] will be utilized.

This framework is an excellent resource for the work, as it will allow the identification of techniques for various application domains in the IT world, especially in the field of ICS, and above all, it will enable the identification of detection and mitigation methods very easily, thereby enhancing the information provided by TAMELESS and allowing the user to take appropriate actions to avoid the problem. Naturally, as stated by the framework's developers, it is not 100% complete, each company faces completely different and unique threats, but it is certainly a good basis for predicting and identifying threats to smart systems.

### 2.3.2 PRISM

**PRISM** is a "probabilistic model checker, a tool for formal modelling and analysis of systems that exhibit random or probabilistic behaviour."[7] It is used in various fields, from communication protocols to systems biology.

The operation of PRISM is based on two main phases:

- The definition of a **model**.
- The definition of the **properties** to be verified regarding the model.

A model is built using the PRISM language, which is simple and state-based. Its main components are the modules and the variables; indeed, a model is composed of one or more modules that can communicate with each other. Each module is characterized by local variables, and their values at any given time define its condition. The system's overall state is then determined by the combined conditions of all modules.[7]

To define a module, the keyword **"module"** is used, followed by the name of the model; to delimit the end of the module, the keyword **"endmodule"** is used. Inside the module, are defined the variables, which are expressed by specifying the name and the initial state. An example is the following:

```
controlBA : bool init false;
```

The behavior of each module is indicated by a set of commands called **"actions"**; they define the change of one or more variables of the model.

An example that is present in the model that will be analyzed later is the following:

```
[compromiseA_rule6] compromisedB & controlBA & spreadBT &
   !safeA -> 1 : (compromisedA' = true) & (restoredA' =
   false);
```

The label inserted inside the "[ ]" is called an "action" and it is the reference to the command in the PRISM language. The variables that follow, namely "compromisedB & controlBA & spreadBT & !safeA", are called **"guard"**. If the condition specified by the guard is satisfied, then the action will be executed, and the update of the variables to the right of "->" will be performed. Each update is assigned a probability, which indicates the actual probability that the variable

changes its state; in this model it is considered 1, since it is desired that the variables to definitely change state if the guard is satisfied.

At the beginning of the PRISM model, it is necessary to specify the type of probabilistic models, since the tool allows one to work with different types. Among these, is possible to find:

- discrete-time and continuous-time Markov chains (DTMCs and CTMCs).
- Markov decision processes (MDPs) and probabilistic automata (PAs).
- probabilistic timed automata (PTAs).
- partially observable MDPs and PTAs (POMDPs and POPTAs).
- interval Markov chains and MDPs (IDTMCs and IMDPs). [7]

The focus is placed on the DTMC (Discrete-Time Markov Chain) model used in this work. The DTMC is a probabilistic model that describes the evolution of a system in discrete time steps. Each state of the system evolves into one or more subsequent states according to transitions that occur with fixed probabilities. Since the security analysis model of TAMELESS [1] is based on formal rules that derive properties starting from the initial state of the system, these rules can be interpreted as discrete transitions from one security state to another, which is perfectly representable by the DTMC probabilistic model.

Once a probabilistic model has been chosen and the various modules defined, PRISM provides tools to evaluate the metrics and correctness of the model through the **properties**. PRISM's property specification language subsumes several well-known probabilistic temporal logics, such as PCTL (Probabilistic Computation Tree Logic) [7], which allows to describe expected behaviors or constraints on the model. The properties enable to verify that the system evolves as expected and to quantify the probability that an event occurs.

There are different types of properties:

- **Qualitative**: They specify behaviors without associating numerical probability values.
- **Quantitative**: They assign probabilistic thresholds, so the property will be true if the threshold is met.

Various operators are available for constructing the different properties:

- **Logical operators:**
  - `&` (AND), `|` (OR), `!` (NOT), to combine conditions.
- **Temporal operators:**
  - `X` (next): indicates the property that must hold in the immediately following state.
  - `F` (eventually): indicates that at some point in the future the property will become true.
  - `G` (globally): indicates that the property must remain true for all future states.
  - `U` (until): expresses that one property must remain true until another property becomes true.
- **Probabilistic operators:**
  - The typical syntax is `P[op threshold] (formula)`, where `op` can be $\geq$, $\leq$, etc.

PRISM has proven to be a fundamental tool for verifying the rules of TAMELESS; it has provided evidence that the rules of the threat model are well-formed and reliable.

### 2.3.3 Neo4J Graph Database

**Neo4J** is an open source graph database developed in Java [8]. It is a non-relational database which, unlike classic relational databases, is highly efficient at handling transactional data, and allows effective data management in contexts where the data are highly interconnected. A NoSQL database of this type uses a graph structure with nodes, edges, and properties to store the data.

The main features of Neo4J can be extrapolated, which are:

- **Horizontal scalability** (in the Enterprise version) : it allows easily adding more nodes to the system. In the Community version, scalability is vertical.

- Neo4j has its own language, created by the company for its query methods – the **Cypher language**.
- The storage is disk-based – through a proprietary file system.
- Its integrity is **ACID** guaranteed.
- It has a very intuitive and very accessible interface. [9]

The language used by Neo4J is quite intuitive and powerful; it easily allows creating nodes, with the addition of properties (as in the example "name" and "type"), and also relationships between nodes. An example found in the TAMELESS code [3] is the following:

```
1 CREATE (attacker:ENTITY {name:"attacker", type:"human"})
    RETURN attacker
```

In addition to the ”CREATE” keyword, Neo4J's Cypher language provides ”MATCH” to return nodes present in the database or ”WHERE” for filtering, as well as keywords that allow more complex operations such as ”COUNT” or ”ORDER BY”.

The database manages transactions implicitly, on an isolated view of the database, through a REST endpoint "tx/commit". The process is as follows:

- The query written in the Cypher language is encapsulated in a `CypherResponse` object which prepares the JSON message to be sent.
- A POST request is made to the "tx/commit" endpoint. At this point, the transaction is opened, the query is executed, and then the commit is performed (i.e., the transaction is closed) in a single operation.
  - If the query is executed successfully, the transaction is automatically committed.
  - If the query fails, the JSON response will contain errors and the transaction will not be committed, thereby rolling back the operation.

Thanks to its nature and functionalities, Neo4J is naturally the best choice for the TAMELESS system, which primarily works with graphs and therefore with highly interconnected data.

### 2.3.4 LLaMA 3.1 with RAG

**LLaMA** (Large Language Model Meta AI) is a family of Large Language Models developed by Meta AI. In this thesis work, the LLaMA 3.1 model was chosen, as it is considered state-of-the-art in the field of LLMs. This version of LLaMA is provided in three variants, which are distinguished by the number of parameters: one version with 8 billion parameters, another with 70 billion, and one with up to 405 billion. This configuration allows for a trade-off between computational resource requirements and model performance. The selected version is the one with 8 billion parameters, as it is sufficient to yield satisfactory results and proved to be the right compromise between the necessary computing resources and efficiency.

LLaMA 3.1 was released under a license that permits its open-source use for both research and commercial purposes, an element that has enabled its widespread use and spreading. Moreover, several key features of the LLM can be identified, such as multilingualism, coding, reasoning, and tool usage. Its efficiency and scalability have also been improved compared to previous models and some competitors, presenting a fairly simple and stable architecture[10] (Figure 2.2).

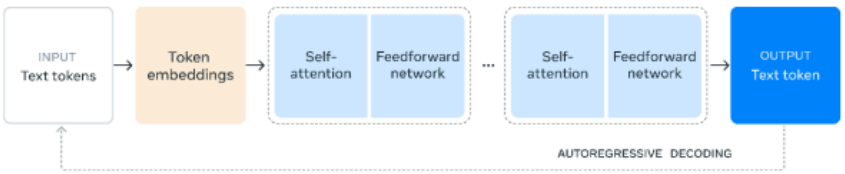


**Figure 2.2:** LLaMA architecture (Adapted from [10]).

Although LLaMA is a reliable model, in this thesis it is used to identify and subsequently validate vulnerabilities present in the MITRE ATT&CK dataset. To reduce errors and hallucinations – situations where the model produces outputs that are semantically and syntactically correct but based on false assumptions or far removed from factual reality – and given the complexity of the required task, the Retrieval-Augmented Generation (RAG) technique was introduced.

RAG is a technique used to improve the output of LLMs by allowing them to generate responses based not only on their training data but also on external authoritative sources, such as the complete dataset of MITRE ATT&CK tactics and techniques for various domains.

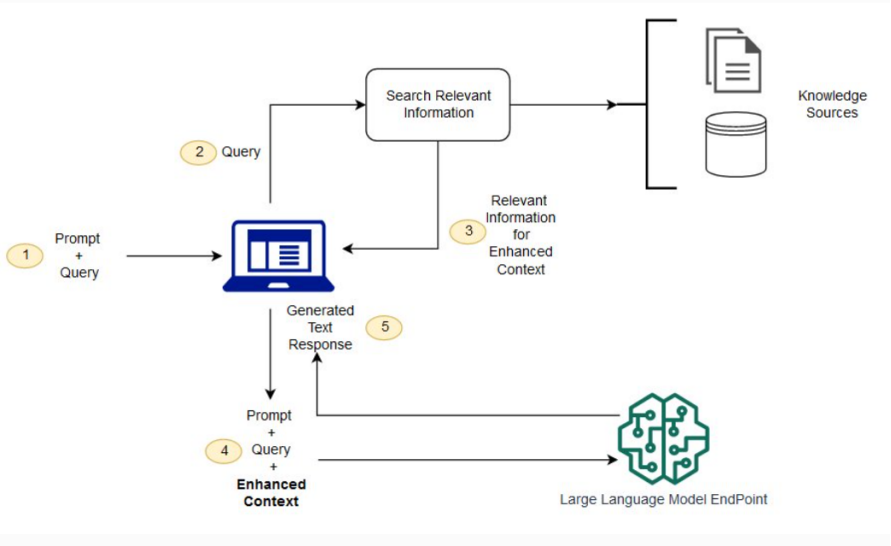


**Figure 2.3:** RAG with LLM workflow. Adapted from AWS [11]

As can be seen in Figure 2.3, the query generated by the user is not immediately sent to the LLM but is first converted into a vector and matched against vector databases to retrieve a number of documents relevant to the query. This information is added to the original query, providing better context to the model and enabling it to return a pertinent answer.

The implementation of RAG in this work is carried out using the LangChain library, a framework for developing applications powered by large language models (LLMs)[12]. It is mainly used to seamlessly integrate language models and vector databases; in this specific case, it is employed to interface with a FAISS index.

FAISS (Facebook AI Similarity Search) is an open-source library developed by Facebook AI Research for similarity search and clustering of high-dimensional vectors[13]. The FAISS index is built from the text embeddings obtained with a SentenceTransformer model. These embeddings represent the threat data extracted from the MITRE ATT&CK dataset. Once created, the index is saved locally and subsequently loaded to execute queries.

In this configuration, given the query, the retriever provided by LangChain performs a search on the vector database to extract information regarding the threats. This information is added to the LLaMA prompt to search for vulnerabilities suitable for a specific node.

The use of this technology has enabled the acquisition of valid data and improve the information regarding the possible threats that can affect the nodes of the system, thereby enriching the formal analysis carried out by TAMELESS.

# Chapter 3

# Introduction to the Case Studies

## 3.1 Examination of the Attack Examples

In this chapter, a selection of the attacks analyzed during the course of the thesis is presented. They were chosen because they align perfectly within the application domain of TAMELESS and represent several scenarios that may occur. Below, all these cases will be analyzed by the new system and the results obtained will be discussed.

### 3.1.1 Water Supply System Hack

On February 5, 2021, Oldsmar, a small town in Florida, was hacked into at their water treatment plant and the hackers attempted to alter the chemical composition of the water[14]. Specifically, they increased the amount of sodium hydroxide from 100 parts per million to 11,100 parts per million, a level that could have made the water highly toxic if consumed by humans.

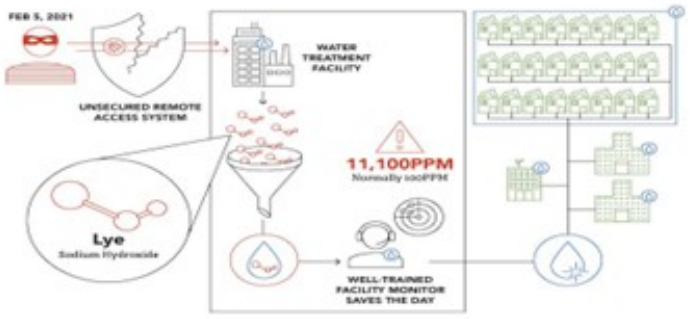


**Figure 3.1:** Water Supply Attack (Adapted from [15]).

The attack was carried out as follows:

1. The attackers presumably obtained the login credentials of the system through a phishing campaign, exploiting the human factor and weaknesses in security training.

2. Using access credentials, the attackers took control of TeamViewer, a software that allows employees to remotely access the system, whose security was not properly managed, due to the lack of multifactor authentication mechanisms and adequate controls.

3. Once inside the system, hackers took control of the plant's IoT-enabled devices. They increased the concentration of sodium hydroxide by changing the settings in the SCADA software.

In the actual incident, the attack was not successfully completed, as an operator noticed the excessive levels of sodium hydroxide in the water and managed to intervene, preventing a fatal outcome. However, the incident clearly highlights how the lack of adequate technical controls and the vulnerability of the human factor can combine to create a hybrid attack capable of simultaneously affecting both the cyber and physical domains.

The case under examination fits perfectly within the ICS category, highlighting how these systems represent critical and sensitive assets, whose security is strictly linked to the safety of people.

### 3.1.2 Cyberattack on the Ukrainian Power Grid

In December 2015 the Ukrainian Power Grid suffered a major multi-front attack, which disconnected several electrical substations, causing power outages for several hours [16].

Attack Steps:

1. The attackers gained access to the entire system through a phishing email containing a malicious document. Once opened, it allowed the installation of malware on the system.

2. The installed malware was originally developed for DDoS attacks but was modified to target SCADA systems (**BlackEnergy**). The malware was used to maintain access to the system and to move laterally within the network.

3. Once the SCADA system was entered by the attackers, they started the malware, whose purpose was to remove all the data from the system. This resulted in the deletion of system files on multiple workstations, which caused it to be difficult for the recovery operations to be executed..

4. Once the power outage was triggered, two additional sub-attacks were executed that further complicated recovery:

    - The UPSs (Uninterruptible Power Supplies) were tampered with and reconfigured so that, once a power outage occurred, the UPS would execute a programmed disconnection of its connected load. This prevented the UPS from fulfilling its role of ensuring continuous power supply during outages, thereby delaying the restoration of electrical service.
    - The attackers launched a DDoS attack on the power companies' customer service phone lines, preventing customers from reporting issues that could have served as an early warning signal.

This case is very interesting not only because it fully involves all three domains of interest, demonstrating how they are interconnected, but also because it is a sophisticated, multi-phase attack. Therefore, it serves as a case study for testing the efficiency of the new system by analyzing its behavior under such attack scenarios.

### 3.1.3 Flicker Frequency Attack

Flicker Frequency Attack [17] is an attack developed in a laboratory that involves IoT devices, specifically smart lights. Researchers have observed that these devices are not adequately protected and that their unconventional use can cause direct harm to people.

This attack is based on a smart LED vulnerability that takes advantage of the way smart LEDs control brightness through pulse-width modulation (PWM). Smart LED bulbs are by default powered and controlled by commands from a dedicated controller through the entire time of their operation. Those commands will, in turn, change the brightness by altering the PWM duty cycle. However, with the Flickering Frequency Attack, the attacker uses API functions that are not mentioned in the documentation to avoid the smoothing methods which work with various frequencies. The process is as follows:

1. The attacker, having gained access to the same network to which the smart LED bulbs and their controller are connected, constructs a UDP command to send to the controller. The controller acts as a gateway between the LAN and the lights and features a control interface accessible from the LAN. This command is designed to force the LED to rapidly switch between two brightness levels that are very close to each other.

2. Since the LED operates by rapidly turning on and off, the attacker precisely controls the timing of these changes by modifying the PWM duty cycle. By setting the off periods to extremely short intervals, the attacker can induce a flickering effect at a frequency that, while imperceptible under normal viewing conditions, is high enough to disrupt normal neural processing in individuals with photosensitive epilepsy.

This attack underscores the importance of protecting these devices, as the potential hazards of IoT devices when security is not adequately considered can be significant. This case was selected because it unites the cyber domain with the human domain, highlighting how the compromise of computer systems can have direct repercussions on daily life and the health of individuals, thereby necessitating an integrated approach to ensure overall resilience against increasingly sophisticated attacks.

### 3.1.4 Stuxnet

In 2010, the Natanz nuclear facility in Iran was attacked by the Stuxnet worm, a highly sophisticated malware designed to sabotage industrial control systems [18]. The attack unfolded in several distinct phases:

1. The attackers introduced Stuxnet into the facility's air-gapped network via infected USB drives, thereby bypassing traditional network isolation measures.

2. Once inside, Stuxnet exploited four zero-day vulnerabilities in Microsoft Windows to escalate privileges and propagate laterally throughout the network.

3. The worm specifically sought out systems running the software used to control the centrifuges. By subtly altering the operational speeds of these centrifuges, Stuxnet induced damaging fluctuations that accelerated wear and eventually led to their failure, all while concealing its actions with advanced rootkit techniques.

4. As a result of this targeted manipulation, the Iranian nuclear enrichment program experienced significant delays. The attack underscored the dangerous potential of cyber weapons to inflict real world physical damage on critical infrastructure.

The cyber-physical hybrid nature of this case is especially important as the attack depicts a very impactful correlation between a cyber attack and physical systems, showing how a well-crafted cyber attack can wreck a physical system.

# Chapter 4

# Formal Verification of the Threat Model

This chapter aims to examine the constraints defined in the threat model [1]. In particular, an analysis will be conducted on the application of the various relationships based on the nature of the different entities that compose them. Then, the model defined on PRISM will be presented, which allows the rules defining the model to be represented, concluding with a discussion of the results obtained.

## 4.1 Limitation of the existing Model

The threat model defined in [1] establishes a syntax based on a series of properties and relations that allow the correct expression of rules, which constitute the functioning of the model. These are syntactically defined, but they have not been formally verified, failing to demonstrate that the system is indeed sound —that is, that every property derivable from its rules is actually true with respect to the model's semantics. Furthermore, it is difficult to prove the correctness of the system because the properties and relations have been abstracted from concrete experiences rather than from formal axioms. In this chapter, the module will be formulated in PRISM to determine whether it is sound or not. In parallel,the nature of the relationships among the various entities will be analyzed, with particular attention to their interaction across the physical, cyber, and human domains. This

study will not only highlight the possible ambiguities arising from the informal interpretation of the relations but will also provide insights for a future semantic formalization.

## 4.2 Analysis of Relationships between Entities

This paragraph aims to delve into the limitations of the threat model presented in [1], focusing in particular on the analysis of the nature of the relationships that bind the various system entities to one another or to potential threats. As mentioned earlier, the model adopts a syntax based on properties and relationships to express the operating rules governing its functioning. However, an in-depth analysis of the correctness of these relationships has not been carried out, and it has not been defined whether they are indeed valid for all cases that may arise in the application context in which the model operates.

In particular, the model defines several relationships between entities and possible threats, which have been abstracted from experience and concrete examples. Among the main relationships, we extract only the basic ones, i.e., those that will later form the high-level relationships. The following definitions are adapted from [1]:

- **Protect (A, B, T):** This relationship expresses that entity $A$ protects $B$ from threat $T$. For example, a firewall ($A$) protects a server ($B$) from cyberattacks ($T$).
- **Monitor (A, B, T):** This indicates that $A$ monitors $B$ to detect the presence of threat $T$. For instance, an IDS system can monitor a server's network traffic to identify intrusions.
- **Contain (A, B):** This specifies that entity $A$ contains $B$ and represents the structure of the system. For example, a room ($A$) contains a server ($B$).
- **Control (A, B):** This expresses that $A$ controls $B$, as in the case where an IT administrator (human) controls access to an IT system.
- **Depend (A, B):** This represents the functional dependency, meaning that the proper functioning of $A$ depends on that of $B$. For example, an application ($A$) may depend on the functioning of a server ($B$).

- **Check (A, B):** This indicates that $A$ verifies whether $B$ is functioning correctly, for example through monitoring systems that detect malfunctions.
- **Replicate (A, B):** This means that $A$ is a replica of $B$ and is used to ensure recovery or operational continuity in the event of a failure.
- **Spread (A, T):** This defines that entity $A$ is capable of propagating threat $T$; for instance, an infected device might spread malware to other network components.
- **Potentially Vulnerable (A, T):** This indicates that entity $A$ may become vulnerable to threat $T$ under certain circumstances, for example if it malfunctions.

The analysis carried out mapped these relationships across the three domains, physical, cyber and human, evaluating all possible combinations and examining their interactions. Thus, for a ternary relationship such as *Protect(A,B,T)*, every combination was evaluated for each element, for instance, *(Human, Cyber, Cyber)*, *(Physical, Physical, Human)*, *(Cyber, Physical, Cyber)*, etc. Each combination was assessed to verify whether the relationship with those elements is indeed meaningful in our model. To do so, a concrete example was sought to demonstrate the logical sense of the relationship.

The results of the analysis highlighted some critical issues:

1. The relationships, although defined in an intuitive manner (e.g., “Contain” indicates the composition of the system), result in different interpretations across the various domains. For example, the “Contain” relationship in the cyber domain translates into a logical dependency, whereas in the physical domain it refers to a spatial relationship.

2. In practice, the analysis verified that all the relationships are applicable independently of the nature of the factors that compose them, meaning that there is an example that demonstrates their logical application. The only exception is the "Contain" relationship, which is not applicable to the Human domain, regardless of the nature of the other end of the relationship. No example was found that does not lead to a logical and semantic ambiguity.

From this analysis it is possible to conclude that, although the threat model offers an intuitive and operationally valid framework for representing the interactions among entities in the physical, cyber, and human domains, it presents some ambiguities that must be carefully considered, particularly as evidenced by the "Contain" relationship.

## 4.3 Verification Methodology with PRISM

The threat model outlined in “A Hybrid Threat Model for Smart Systems” [1] describes a set of rules that are used to determine the state of various entities, actually identifying derived properties. It is possible to distinguish these rules into **Basic Derivation Rules** and **Specific Derivation Rules**. An overview of the defined rules will first be presented, and then a formal analysis will be addressed with PRISM.

"Basic derivation rules state that if a property is assumed true, then naturally it can be derived to be true (Rules 1-3). Furthermore, these rules express the transitivity of relations such as Replicate and Depend."[1]

$$\alpha\text{Comp}(A,T) \;\rightarrow\; \kappa\text{Comp}(A,T) \tag{1}$$

$$\alpha\text{Vul}(A,T) \;\rightarrow\; \kappa\text{Vul}(A,T) \tag{2}$$

$$\alpha\text{Malfun}(A) \;\rightarrow\; \kappa\text{Malfun}(A) \tag{3}$$

$$\text{Replicate}(A,B) \;\wedge\; \text{Replicate}(B,C) \;\rightarrow\; \text{Replicate}(A,C) \tag{4}$$

$$\text{Depend}(A,B) \;\wedge\; \text{Depend}(B,C) \;\rightarrow\; \text{Depend}(A,C) \tag{5}$$

These rules act as basic rules or axioms, establishing that if a property is assumed,it can also be derived, while formalizing the transitivity of certain relationships. Being inherently deterministic, they can be excluded in a formal analysis with PRISM, as their effect is intrinsic to the system and does not influence probabilistic evolution.

Focus is now on the Specific Derivation Rules, which represent the “reasoning about how threats can compromise different entities and propagate through the system” [1]. The analysis will be conducted on the rules extracted from [1].

- **Rule 6**: A can be compromised by threat T when A is not safe from T and an entity B, which controls A, can be compromised and spread threat T.

- **Rule 7**: A can be compromised by threat T when A is not safe from T, and A is connected to B through C, B can be compromised and spreads T, and either C can be compromised or C is not protected against T.

- **Rule 8**: A can be compromised by threat T when A is not safe from T, and either A contains B or is contained in B,and B can be compromised and spread T.

- **Rule 9**: specifies that an entity A, compromised by threat T can malfunction.

- **Rule 10**: specifies that A can malfunction when A depends on B and B malfunctions.

- **Rule 11**: states that when A malfunctions, and A is potentially vulnerable to threat T, then A can be vulnerable to T.

- **Rule 12**: states that when A can be compromised by threat T and A is monitored for threat T by some entity that is not compromised, then T can be detected for A.

- **Rule 13**: states that when threat T can be detected for A and A has been replicated, then A can be restored.

- **Rule 14**: states that when A can malfunction and is checked then A can be fixed.

[1] Specific derivation rules are represented formally, as it will be the syntax used to represent them on PRISM.

$$\text{Control}(B, A) \wedge \kappa\text{Comp}(B) \wedge \text{Spread}(B, T) \wedge \neg\text{Safe}(A, T) \rightarrow \kappa\text{Comp}(A, T) \tag{6}$$

$$\text{Connect}(C, B, A) \wedge \kappa\text{Comp}(B) \wedge \text{Spread}(B, T) \wedge \neg\text{Safe}(A, T) \wedge (\kappa\text{Comp}(C) \vee \neg\text{Def}(C, T)) \rightarrow \kappa\text{Comp}(A, T) \tag{7}$$

$$(\text{Contain}(B, A) \vee \text{Contain}(A, B)) \wedge \kappa\text{Comp}(B) \wedge \text{Spread}(B, T) \wedge \neg\text{Safe}(A, T) \rightarrow \kappa\text{Comp}(A, T) \tag{8}$$

$$\kappa\text{Comp}(A, T) \rightarrow \kappa\text{Malfun}(A) \tag{9}$$

$$\text{Depend}(A, B) \wedge \kappa\text{Malfun}(B) \rightarrow \kappa\text{Malfun}(A) \tag{10}$$

$$\kappa\text{Malfun}(A) \wedge \text{PotentiallyVul}(A, T) \rightarrow \kappa\text{Vul}(A, T) \tag{11}$$

$$\kappa\text{Comp}(A, T) \wedge \text{Mon}(A, T) \rightarrow \kappa\text{Det}(A, T) \tag{12}$$

$$\kappa\text{Det}(A, T) \wedge \text{Rep}(A) \rightarrow \kappa\text{Rest}(A) \tag{13}$$

$$\kappa\text{Malfun}(A) \wedge \text{Che}(A) \rightarrow \kappa\text{Fix}(A) \tag{14}$$

The model, defined within the formal verification framework, is now presented. As mentioned in Chapter 2 the DTMC probabilistic model will be used.

First,the form containing the local variables is defined. In this case, they all represent properties and relationships with their respective initial values, chosen so as not to trigger any rules.

**Listing 4.1:** PRISM Module Variables

```
1  compromisedA    : bool init false;
2  compromisedB    : bool init false;
3  compromisedC    : bool init false;
4  controlBA       : bool init false;
5  spreadBT        : bool init false;
6  safeA           : bool init true;
7  defC            : bool init true;
8  malfunctionA    : bool init false;
9  malfunctionB    : bool init false;
10 vulnerableA     : bool init false;
```

```
restoredA        : bool init false;
dependentAB      : bool init false;
detectedA        : bool init false;
monitoredA       : bool init false;
replicaA         : bool init false;
checkA           : bool init false;
connectCBA       : bool init false;
protectC         : bool init true;
containBA        : bool init false;
potentiallyVulA : bool init false;
```

Once the local variables are represented, the actions that will change their states are defined; then, referring back to the formal rule writing seen earlier, the actions that are needed to activate each rule are represented. Let's consider at an example for rule 6:

**Listing 4.2:** PRISM role 6

```
[compromiseB] !compromisedB -> 1 : (compromisedB' = true);
[controlBA] !controlBA -> 1 : (controlBA' = true);
[spreadBT] !spreadBT -> 1 : (spreadBT' = true);
[loseSafety] safeA -> 1 : (safeA' = false);
[compromiseA_rule6] compromisedB & controlBA & spreadBT &
   !safeA ->  1 : (compromisedA' = true) & (restoredA' =
   false);
```

As seen above, when the condition specified by the guard is met, the action is performed, leading to the state change of the variables specified to the right of the action. It is intended to emphasize that the execution of PRISM is not sequential like that of a typical imperative program, but during model checking, PRISM explores the entire state space in a systematic way, to evaluate the various properties specified. Thus, in this example, all variables that are part of rule 6 have been explicated consecutively just to improve readability and understanding of the module.

In addition, each rules will also have an associated reset action to return some variables, used temporarily for rule activation, to their default value once the rule has been executed, maintaining state consistency and preventing undesirable

behavior due to flags that persist beyond their initial purpose. An example is as follows :

**Listing 4.3:** PRISM reset role 6

```
[resetRule6] compromisedA & (compromisedB | controlBA |
  spreadBT | !safeA) ->  1 : (controlBA' = false);
```

After defining all the rules with the corresponding reset actions, the module is concluded by introducing the introduction of the **formula**, an operator made available by the model checker that avoid duplicate code. To define it, a label followed by an expression is written after the keyword formula, ensuring a more compact notation for properties. An example is the following:

**Listing 4.4:** PRISM formula role 6

```
formula role6 = compromisedB & controlBA & spreadBT &
  !safeA;
```

In this way, a formula is defined for all the rules presented so far.

## 4.4 Results and Discussion of the Validation

After the creation of the model, the properties of interest for formal verification must be defined.
First, it was verified that each rule is activated without creating a logical conflict with other rules. To achieve this, the model must verify that each rule, once activated during a transition, consistently returns ($\mathbf{P >= 1}$) the expected result at the next transition. The properties script is as follows :

**Listing 4.5:** PRISM properties

```
P>=1 [ F ( role6 & (X compromisedA))]

P>=1 [ F ( role7 & (X compromisedA ))]

P>=1 [ F ( role8 & (X compromisedA ))]
```

```
6
7  P>=1 [ F ( role9 & (X malfunctionA))]
8
9  P>=1 [ F ( role10 & (X malfunctionA))]
10
11 P>=1 [ F ( role11 & (X vulnerableA))]
12
13 P>=1 [ F ( role12 & ( X detectedA))]
14
15 P>=1 [ F ( role13 & (X restoredA & !compromisedA))]
16
17 P>=1 [ F ( role14 & (X fixedA & !compromisedA))]
```

The **F** operator is used to verify that the content of the parentheses is validated at a certain time frame. In the PRISM language, F means 'eventually' and returns true, if at any transition, the content of its parentheses is true. For each rule, verification is performed to ensure that its conditions are eventually checked without interference from other rules. To simplify notation, the previously defined 'formula' is applied. In addition to verifying their activation, it is necessary to check that the result is correct. To achieve this, the AND operator is used. Furthermore, to ensure that the result is recognized by the system at the next transition, the **X** operator, known as 'next', is applied. This operator returns true if the result is visible in the subsequent state.
For rules 13 and 14, in addition to verifying the outcome of the actions, verification confirmed that the implication of the rules is satisfied, meaning that a restored or fixed entity cannot be compromised again. This ensures the liveness of the system and its resilience.

To better verify the resilience of the model and ensure that it does not become locked in a given state, it was defined an even more thorough property. This property ensures that whenever "compromisedA" becomes true, it eventually (F) succeeds to restore the entity and at the next state it must not be compromised again. Here, the operator G (Globally) is introduced which is only valid if its condition is valid for all states along the model path. This is the properties:

```
P>=1 [ G ( compromisedA => (F restoredA & (X
   !compromisedA)) ) ]
```

In addition, it is necessary to verify that the system can deal with multiple cycles of compromise and restoration, ensuring that there is a nonzero probability that, if the system is compromised, it can be restored to be compromised again.

```
P>0 [ F ( compromisedA & (F (restoredA & (F compromisedA))
   )) ]
```

However, at the same time is also essential to verify that an entity can pass from a compromised state to a restored one, without immediately return to being compromised, so recovering and maintaining its safe state.

```
P>0 [ (compromisedA) => (F (restoredA &  (X !compromisedA)))
```

The analysis continues by verifying that an compromised state cannot exist without the actual presence of the conditions, so at least one of the rules leading to that state must be verified.

```
P>=1 [ G ( !role6 & !role7 & !role8 & !role9 =>
   !compromisedA ) ]
```

It is also intended to emphasize the "Detect" relationship, going to verify that if an entity is compromised and also monitored, then the system can reliably detect the threat in any path state.

```
P>=1 [ G ( compromisedA & monitoredA =>( F detectedA) ) ]
```

The formal analysis is concluded by ensuring model consistency, preventing state variables from remaining activated and allowing the system to return to a safe state.

```
P>=1 [ G ( restoredA => (compromisedA = false) ) ]

P>=1 [ G ( fixedA =>  ( malfunctionA = false) ) ]
```

Once all properties have been verified, it is possible to say that the formal analysis has shown that the system is robust, consistent, and capable of handling critical transitions. The model succeeds in ensuring the correct activation of the rules, without any conflict between them. It also proves to support multiple cycles of compromise and recovery by not creating deadlocks or loops in the system, and prevents unwarranted transitions by ensuring that various states are verified when supported by the correct conditions.

# Chapter 5

# Evolution and Technological Integration of TAMELESS

## 5.1 Compatibility with Neo4j

TAMELESS code remained essentially unchanged from the original version; the modifications that have been made were only necessary to update it to the new versions of Neo4J, making TAMELESS a tool that is fully usable and up-to-date. For this reason, the main changes have been made to the file **Neo4jDB** located at "TAMELESS/src/it/polito/dp2/TAMELESS/sol/db/Neo4jDB.java", which acts as an "Orchestrator" between the service layer and the DAO (Data Access Object), and to the file **MyNeo4jClient** located at "TAMELESS/src/it/polito/dp2/TAMELESS/sol/db/MyNeo4jClient.java", which is the DAO itself, directly handling communication with the database.

The old version of Neo4J managed communication by providing a REST API to perform CRUD operations. The base endpoint was "http://localhost:7474/db", and in addition to that, there were other endpoints available for data management. Among the most common there are:

- **/node** – used for managing nodes, allowing for read, write, update, and delete operations.
- **/relationships** – used for managing relationships between nodes.

- **/schema** – used to manage the database schemas, such as uniqueness constraints and indexes.
- **/transaction** – used to handle multiple transactions in a single request. To use it, the query had to be written using the Cypher language.

For the various endpoints, CRUD operations were carried out using the corresponding HTTP calls (GET, POST, PUT, DELETE). In practice, to perform a simple operation such as reading a node, a GET request can be directly issue to the "/node" endpoint, specifying the node's id. Neo4J would directly handle the transaction. If a more complex operation is required, it is necessary to manually open a transaction by making a request to the "/transaction" endpoint to make the operation atomic. In such cases, managing rollbacks for errors or failures is also necessary. Furthermore, the old version of Neo4J did not manage node authentication, a mechanism that makes communication with the DB more secure by ensuring that only authorized users can perform operations on the database.

The new versions of Neo4J, from 4.x onward, have brought several improvements. The first among these is the implementation of authentication to connect to the database. This is carried out through the register function of the Client object, which performs Basic Authentication provided by the HTTP protocol. Therefore, the HTTP request will include an "Authorization" header whose value is the "credentials", that is, the combination of the username and password separated by a ":" and encoded in Base64.

```
client.register((ClientRequestFilter) requestContext -> {
    String credentials = "neo4j:tameless";
    String encodedCredentials =
    Base64.getEncoder().
        encodeToString(credentials.getBytes());
    requestContext.getHeaders().add("Authorization", "Basic
   " + encodedCredentials);
});
```

In addition, communication still takes place through HTTP requests, but the base path of the endpoint has been changed to "http://localhost:7474/db/neo4j",

and all the other endpoints have been removed and replaced by the new endpoint: "/tx/commit". This change still allows a Cypher query to be executed in an atomic transaction, autonomously handling commit and rollback, and improving the efficiency of the database, albeit at the cost of making the code a bit more complex.

Thus, in the file "MyNeo4JClient.java" all the modifications described above have been applied, leading to changes in several functions to make the system compatible with the new versions. An example of a change is as follows, moving from this version:

```
Response response = target.path("node").path(id.toString())
                    .request()
                    .delete();
```

to this one:

```
String query = "MATCH (n) WHERE id(n) = " + id.longValue() +
                " DETACH DELETE n";

CypherRequest request = new CypherRequest();
Statements stat = new Statements();
stat.setStatement(query);
stat.getResultDataContents().add("REST");
request.getStatements().add(stat);

CypherResponse response = target.path("tx").path("commit")
            .request(MediaType.APPLICATION_JSON_TYPE)
            .post(Entity.json(request),CypherResponse.class);
```

Regarding the file "Neo4JDB.java", small modifications have been applied to some functions to better adapt them to the changes made in the underlying layer. Finally, one last modification was made to TAMELESS: a new endpoint "http://localhost:8080/TAMELESS/rest/system/all" has been added. Through a DELETE call is possible to remove all nodes and relationships present in the

graph by leveraging the delete functions already present in the system. This change was added to automate analyses through the GUI, allowing the system to switch automatically from one example to another.

## 5.2 Development of the GUI

To use TAMELESS and perform a threat analysis, it is required to use the APIs provided by the tool. This facilitates the integration with with other systems, but at the same time it does not make it easily accessible and usable by most users, especially because the results are currently returned in JSON format, leading to a not intuitive interpretation. For these reasons, it was decided to integrate a GUI into the system: a fairly simple but effective interface that allows for the easy input of the graph to be analyzed and the clear visualization of the results.

To build the interface, the React framework[19] was used, which simplifies the development of components and data management. To improve the visual appearance, Tailwind CSS[20] was employed, a framework that facilitates the rapid and intuitive construction of functional graphical interfaces.

The main components of the graphical interface are two: **GraphEditor.jsx**, which allows the insertion of the graph, the security properties, and the initiation of the analysis; and **GraphResults.jsx**, which enables the visualization of the results, with the option to view only those obtained through formal analysis or to integrate them with those obtained from LLaMA.

Regarding **GraphEditor.jsx**, a section divided into three parts has been included:

1. A form for inserting nodes that features a box to type the node's label and a select for the type (Human, Cyber, Physical);

2. A second form for inserting edges where it is possible to select, among the nodes present in the graph, the source node and the target node;

3. A box for loading the test cases to simplify the testing process. The data representing the attacks discussed in Chapter 3 are loaded through an appropriate function (*loadData.js*), allowing the immediate visualization of the graph (Figure 5.1).

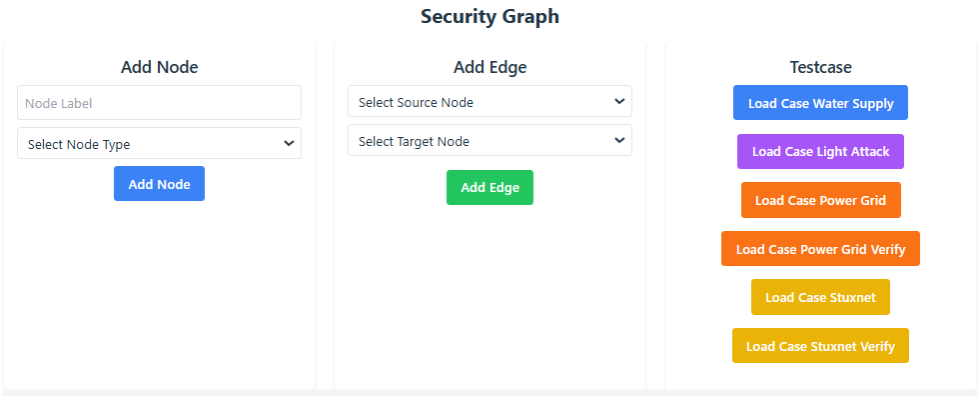


**Figure 5.1:** GraphEditor's form.

The graph visualization was performed using the Cytoscape[21] library, which allows transforming data into an interactive graphical representation, dynamically arranging the nodes and providing the possibility to automatically rearrange them using the "Rearrange Graph" button (Figure 5.2).

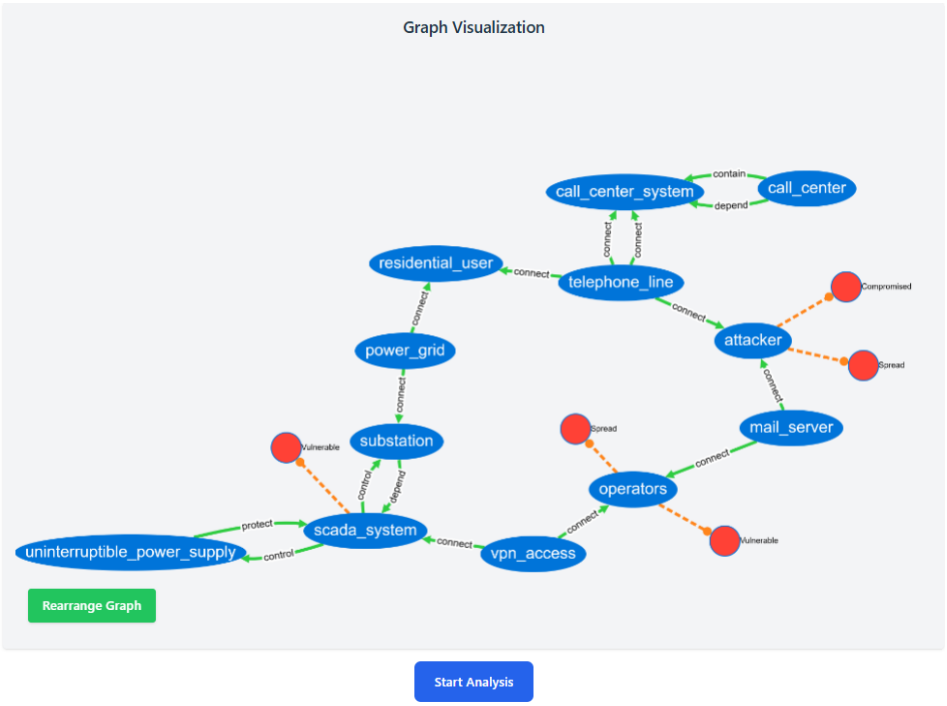


**Figure 5.2:** Cryptoscape graph visualization (Ukrainian Power Grid).

The graph presented here represents the **Cyberattack on the Ukrainian Power Grid** discussed previously. The **blue nodes** represent the system's entities; the edges represent the relationships, denoted by a label; and the security properties are represented by the **red nodes**. The nodes representing the threats have been intentionally excluded, as it was preferred to simplify the interface in this initial phase.

To add or modify a relationship, simply click on the corresponding edge. Once done, a form will open allowing the selection of the desired relationship to add and specify either the threat or a third node, based on the selected type of relationship (Figure 5.3). If, instead, the user wants to add or modify the properties of a node, simply click on it; a second form will open that allowing the view of the various properties already present and to delete or add new ones. From the same form, it is also possible to delete the node or save the newly added relationship; in this case, a new node will appear in the graph visualization (Figure 5.4).

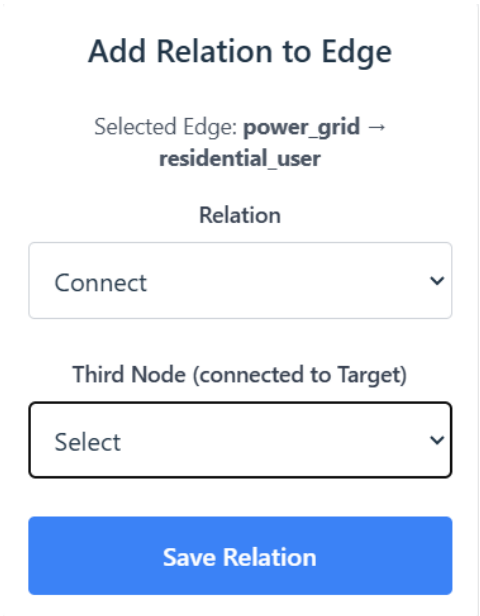


**Figure 5.3:** Graph relation form

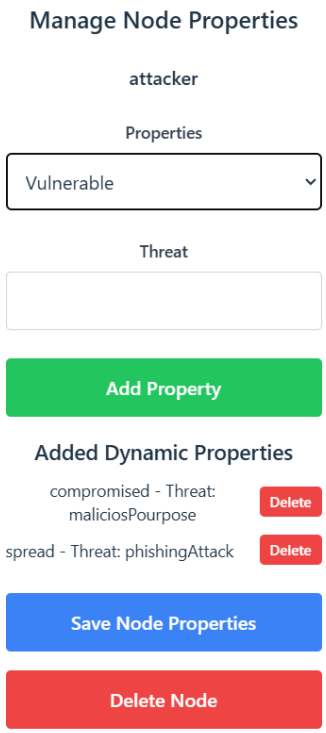


**Figure 5.4:** Security Properties form

Once the graph is defined, the analysis is started by clicking the **"Start Analysis"** button. Here, the data will be sent to the orchestrator (*graph_processor.py*), whose functioning will be presented in the next chapter, and a status bar will appear that informs the user of the analysis progress. This is done using the polling technique: every 5 seconds, the component calls the **"/get_results"** endpoint of the orchestrator to obtain information on the analysis progress. This mainly refers to the analysis performed with the LLM, as it is the most resource-intensive and time consuming; therefore, information on the overall system completion percentage and progress details for a specific node will be displayed.

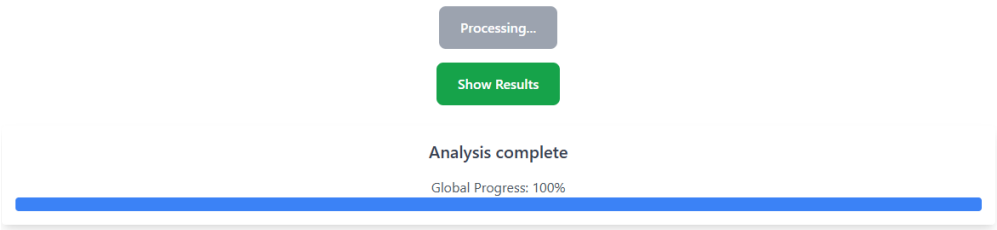


**Figure 5.5:** Analysis Status bar.

Once the analysis is completed, by clicking the **"Show Results"** button the second component **GraphResults.jsx** will be invoked. It will contact the */get_results* endpoint of the orchestrator, which will return the data containing the results of both analyses performed in parallel. Initially, only the results obtained from the formal analysis will be displayed, showing all the system nodes and the respective threats declared in the properties earlier. Furthermore, each node has a security status represented by a color, and a legend is provided to associate each color with its meaning.


**Figure 5.6:** Ukrainian Power Grid TAMELESS results.

Below this screen, there is another button that allows the visualization of the results obtained with ATT&CK. Once pressed, the graph will change; in fact, it will be filtered to contain all the entities present in both sets of results, thereby reducing the "noise" caused by the other nodes and highlighting the main attack path. However, it is always possible to return to the main view to obtain a complete picture of the entities present in the system. Additionally, some metrics are reported that might be of interest to the end user and that emphasize the contribution of the model. The main metrics chosen are three:

1. **Graph reduction** obtained when merging the results;

2. The number of nodes added by ATT&CK, which have been associated with a threat but were not identified by the formal analysis;

3. The names of these entities, allowing for immediate identification and verification.

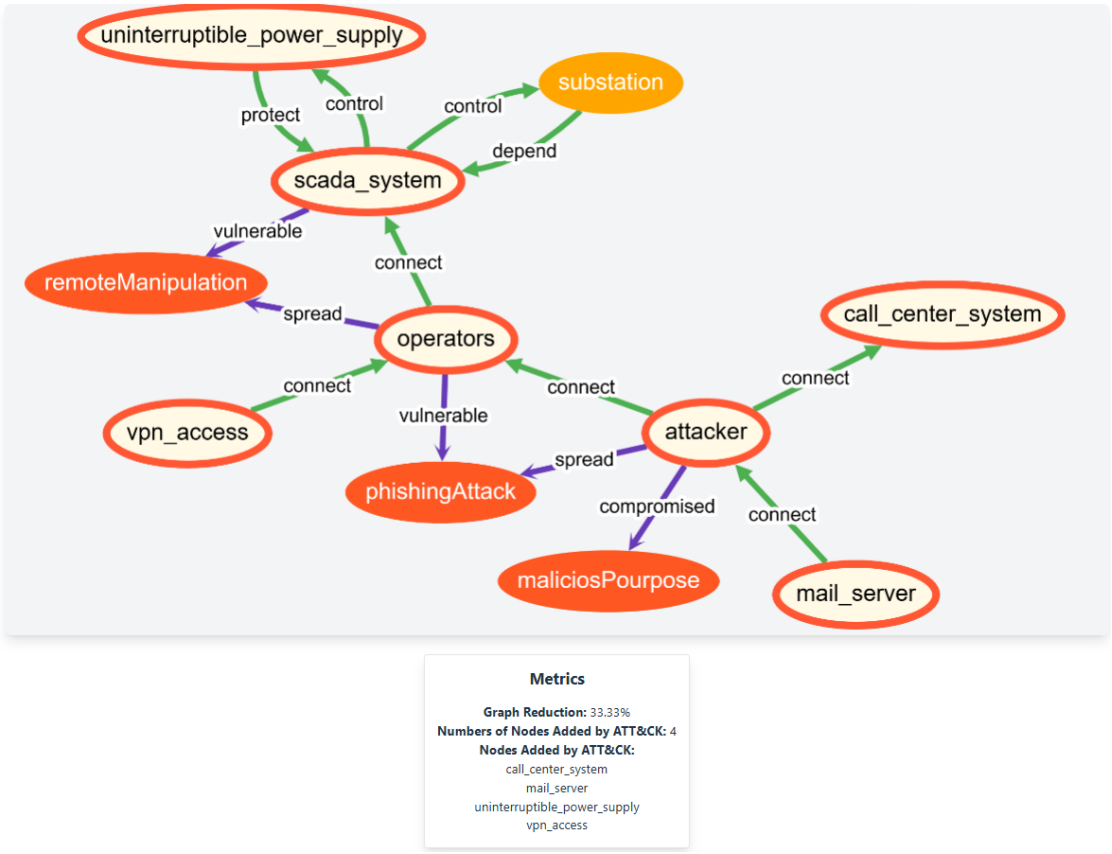


**Figure 5.7:** Ukrainian Power Grid Combined Results.

Furthermore, it is possible to click on each node to view the tactics and techniques identified by the LLM. As shown in Figure 5.8, several pieces of information are provided:

- The probability that the node is compromised.
- The technique along with its corresponding ID.
- The probability related to the exploit of that specific technique, combining the information from MITRE with that from TAMELESS.
- A description taken from the MITRE dataset.

- The reason provided by the model to justify the exploitability of the technique on that node.
- Detection techniques recommended by ATT&CK to best identify the threat.
- A direct link to the MITRE website, to have a complete overview of the identified technique.
- A list of all possible mitigations to be implemented.

This data is provided to give the user a complete overview, showing not only the possible threats that can affect the node, but also various countermeasures, allowing for the best possible defense of the system. The calculation of metrics, such as probability and risk, will be discussed in more detail in the next chapter.


**Figure 5.8:** LLaMA example results .

# Chapter 6

# LLM-Enhanced Threat Analysis

In this chapter, the main modification made to TAMELESS is addressed, namely the change in the flow concerning the threat analysis process, introducing a parallel process executed with LLaMA that will identify possible threats for each entity in the system. Therefore, the new architecture, the integration with the results of TAMELESS, and an analysis of the probability and risk metrics that have been added will be presented.

## 6.1 MITRE ATT&CK via LLAMA 3.1 with RAG

During the study regarding the rules of TAMELESS and the verification of their applicability to the identified case studies, an important feature of the tool emerged, namely its generality. TAMELESS is capable of representing many scenarios, adapting well to various situations and examples, successfully identifying the security states of different entities. Given this characteristic, the work was extended to provide additional information, enabling the user not only to determine whether a node may be compromised but also identify the actual threats that could endanger the node.

For this reason, it was decided to extract data on the most common threats from a dataset, and after research, the ideal candidate for the analysis scenarios of

TAMELESS turned out to be MITRE ATT&CK. Once the framework for extracting threats was identified, it was necessary to determine a technique that would allow to retrieve these vulnerabilities based on the node being analyzed. Additionally, it was crucial that this threat could be exploited considering the entire system in which the node is embedded.

As a result, an LLM was selected for this task, enabling it to "reason" about the node: first by identifying possible threats and then, in a second phase (using another prompt), validating them based on the entire system and the node's connections.

The structure of the new workflow is presented here:

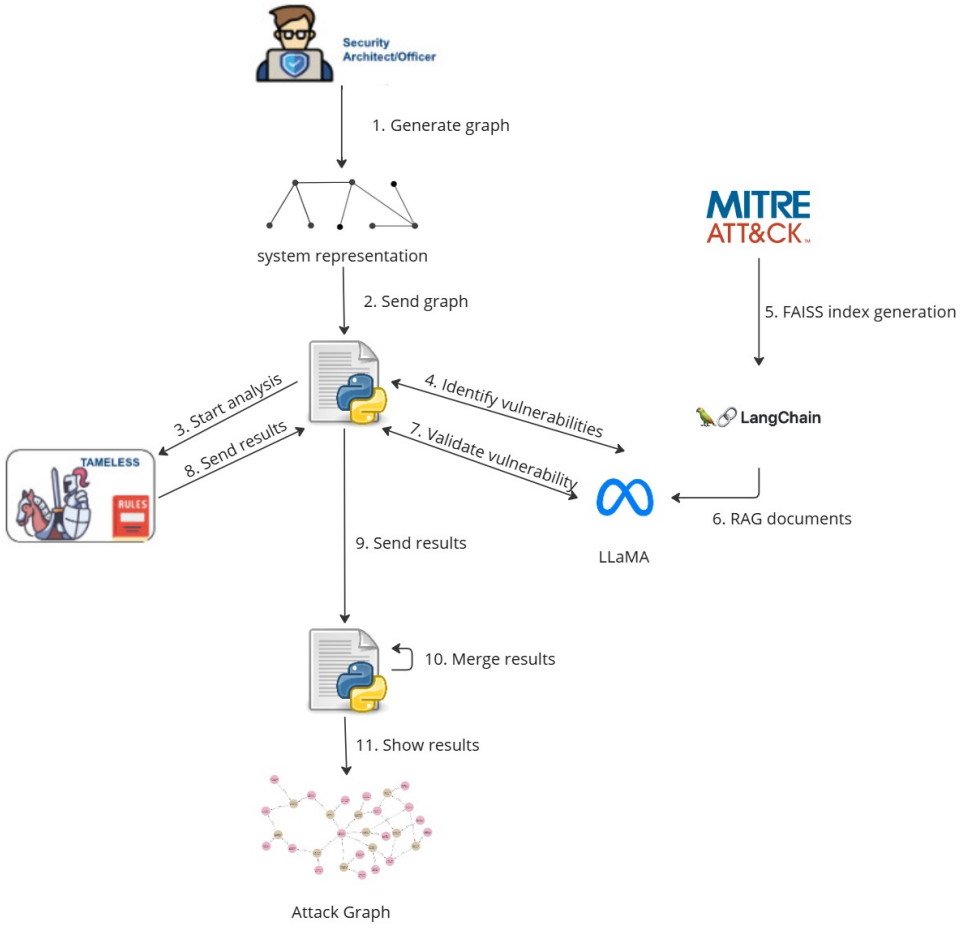


**Figure 6.1:** TAMELESS plus LLaMA workflow.

The Security Architect, through the use of the GUI, inputs the data representing the system to be analyzed, including the relationships and properties of the various entities. Once the analysis begins, the graph is sent to an orchestrator, which, after obtaining the data, initiates two parallel analyses. First, it forwards the graph information to TAMELESS, which conducts its analysis as it did previously. Subsequently, the same data is leveraged for an analysis with LLaMA. For each node contained in the graph, a request is sent to LLaMA to identify possible threats. Before the request reaches the model, it is enriched via RAG with information about threats contained in the dataset.

Once the list containing the MITRE tactics and techniques is obtained, it is iterated to send each individual threat to a second endpoint of the LLaMA module, which this time will validate it considering the entire system.

When both the TAMELESS analysis and the LLaMA analysis are completed, the results are sent directly to the GUI, which performs a merge, allowing users to view the formal result from TAMELESS or a smaller graph containing only the unified results from both analyses.

This provides an overview of how the new analysis process functions; now, the operation of the various components will be presented in more detail.

The user enters the data to form the graph, generating a JSON file containing:

1. The list of nodes with their respective *id*, *label*, and *type*;
2. all the edges of the graph, which indicate the source and target node, the name of the relationship, and the *category* that specifies the type of relationship;
3. a second object called `analysisData` containing the key data for the two analysis processes:
    - **SecurityProperties**, which indicate the information that defines a property of an entity.
    - **Relations**, a list containing all the relationships between entities and threats defined in the TAMELESS format.

```
{
  "nodes": [
    {
      "id": "attacker",
      "label": "attacker",
      "type": "human"
    },
    ...
  ],
  "edges": [
    {
      "source": "mail_server",
      "target": "attacker",
      "relation": "connect",
      "category": "Relation3Entities"
    },
    ...
  ],
  "analysisData": {
    "securityProperties": [
      {
        "nodeId": "attacker",
        "property": "compromised",
        "threat": "maliciosPourpose"
      },
     ...
    ],
    "relations": [
      {
        "source": "mail_server",
        "target": "attacker",
        "relation": "connect",
        "thirdNode": "operators",
        "threat": null,
        "category": "Relation3Entities"
```

```
36        },
37        ...
38      ] } }
```

These data are sent to the dispatcher called *"graph_processor.py"*, a Flask server that exposes three endpoints:

- **/process_graph** handles receiving the data and starting the parallel analysis; it splits the data to perform the analysis with TAMELESS and opens a thread to perform the analysis with the LLM.
- **/progress** is contacted by the frontend via polling to update the loading bar.
- **/get_results** returns a JSON object containing the results of the two analyses.

### 6.1.1 TAMELESS analysis

The analysis with TAMELESS is managed by two components: *tameless_analysis.py* and *tameless_handler.py*.

Both provide a class, respectively `TamelessAnalysis` and `TamelessHandler`. The first one creates a `TamelessHandler` object and defines methods to analyze the data structure mentioned earlier and to create a cache. This will be an element that contains all the information from the TAMELESS analysis, i.e., all the entities, threats, properties, and relations with their respective IDs from the Neo4J database, and, at the end of the analysis, also the results. This was done because TAMELESS returns a URI to be recalled to obtain detailed information about the various nodes, as well as the IDs of the elements involved in the result, typically the ID of one or more entities and the ID of the involved threat. To avoid an additional GET call, it was preferred to save the data locally so as to have a direct association between IDs and the various information of the node.

The `TamelessHandler` object, on the other hand, provides a series of methods that enable communication with the tool's API; so by creating XML objects, it allows for the insertion of entities, threats, properties, and relations. Naturally, it also allows requesting the computation of the results, which, as mentioned, will then be saved in the cache.

### 6.1.2 LLaMA analysis

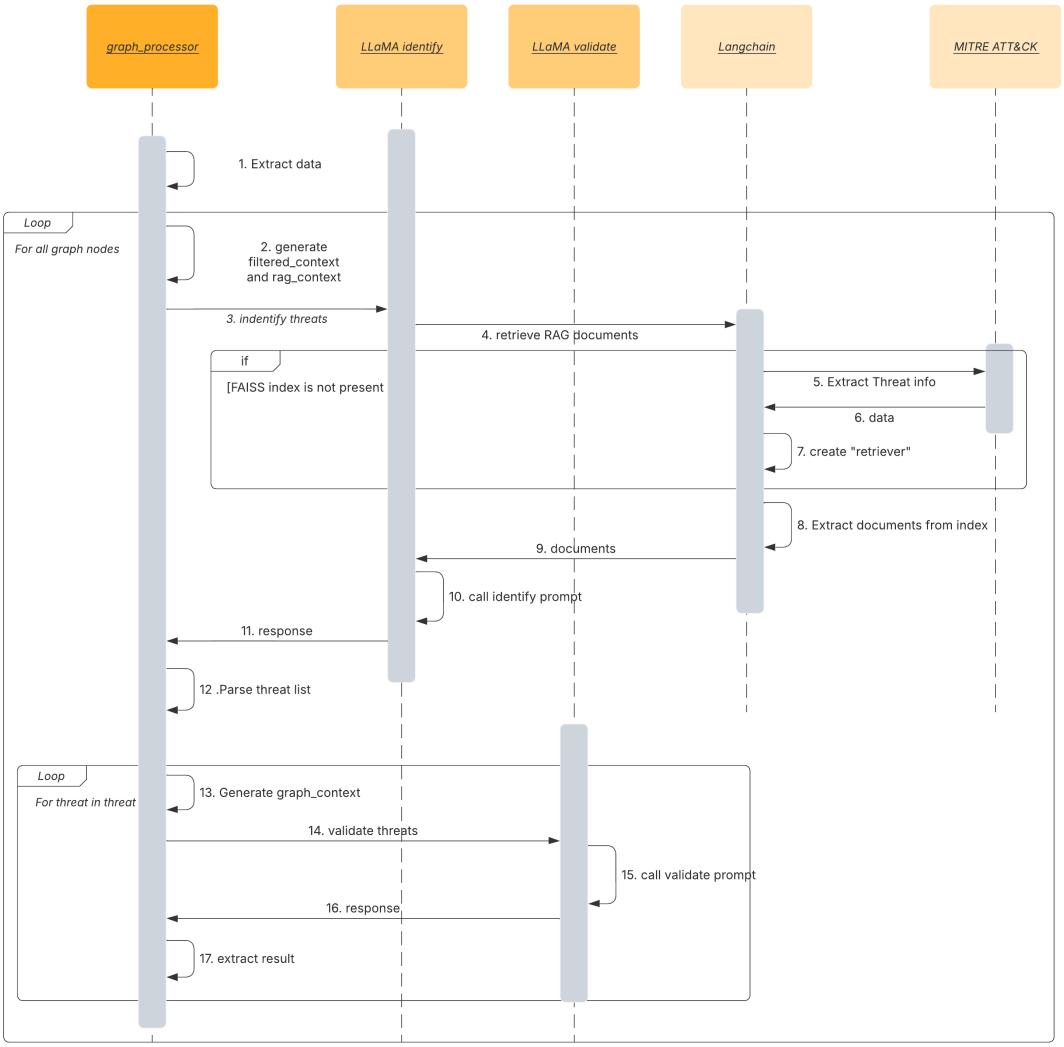


**Figure 6.2:** LLaMA flow diagram.

The analysis with LLaMA involves the same *graph_processor.py* and the module *llama_vulnerability_analyzer.py*, which is responsible for loading the model and defining the prompts, another Flask server that exposes two endpoints: **"/identify"** and **"/validate"**. The model is downloaded using the HuggingFace library [22], which allows loading the model into memory without saving it locally. Despite this, to make the process lighter and avoid saturating computational resources, a

4-bit quantization of the model has been added, using the `BitsAndBytesConfig` library. In this way, the model's weights are represented in 4 bits, instead of the usual 16 or 32 bits. However, this entails a loss of accuracy in the representation of the model's weights, which is nevertheless minimized through specific algorithms.

To enable a parallel analysis, and because LLM execution takes too long to keep an HTTP connection open, its execution has been entrusted to a separate thread that carries out the operation. The thread runs a function that, by extracting the data from the JSON, recreates a graph structure using the *networkx* library. Once the graph is defined, it iterates through it to process each individual node. For every node, a second function is called to define a **partial context**,– a string describing all the elements of the system including the relationships and security properties related to that specific node. Additionally, a *rag_context* is generated, which contains only all the properties of that node. Then the node information, together with the partial context, is sent to the "/identify" endpoint which, once it receives the data, first checks whether the FAISS Index has been created or not. If the index is not present, it extracts the data from the ATT&CK dataset using the Sentence Transformer library, specifically the model *all-MiniLM-L6-v2* [23]. It then performs data embedding, by taking strings (tactics, techniques, descriptions, and MITRE ATT&CK IDs) and transforming them into a series of numberical vector, which represents the semantic features of the text. This allows comparing the meaning of the texts mathematically and performing semantic searches, finding related content.

Once the index is created, the langchain library allows creating an object called *"retriever"* that, given a query as input, uses the FAISS index to retrieve the most relevant information or documents. The query received by the retriever is as follows:

```
1     query = f"""
2       Node Name: {node_name}
3       Node Type: {node_type}
4       Context :{rag_context}
5     """
```

Using this information, the retriever selects a number of documents that are returned to the function and added as Context to the model's prompt. At this point, all the node information, the system context, and the rag context are passed to LLaMA.

The prompt, which can be found in Appendix A.1, is structured as follows. In addition, several prompt engineering techniques are applied:

1. Through the Role-playing technique [24], the model is asked to take on the role of a "cybersecurity expert," specifying its main goal, so that the responses are generated in a specific professional context.

2. Detailed contextual information, relevant to the task, is provided, ensuring that the model has the necessary data to produce pertinent answers.

3. The task is made explicit, but through the "Chain-of-Thought Prompting" technique [25], it is broken down into smaller, more manageable steps, guiding the model through detailed instructions.

4. Based on the supplied of the node type, the model gets a set of conditional instructions that are provided. These are meant to guide it in finding the most suitable methods based on the node being analyzed, by providing the explanations and examples.

5. The model is forced to generate a specific output format, through Output formatting [26]. In this case, a strict JSON format is requested, and an example is provided to ensure consistency and ease of response processing.

6. Finally, specific constraints for output generation have been inserted, helping the model maintain a focused and relevant answer.

The model will return a raw response that includes the entire prompt and then its answer, which will be a JSON containing the following fields:

```
{
    "response": [
      {{
        "Tactic": {{tactic}},
        "Technique": {{technique}} - {{MITRE ID}},
        "Description": {{Official MITRE Description}}
      }},
      ...
    ]
}
```

The response will contain the entire prompt; for this reason, a flag like *"«<END PROMPT»>"* has been introduced at the end, which is used to manipulate the text through regex and obtain the list of threats. This list is iterated, and for each threat, a string containing all the nodes in the system, all the relationships, and the properties of the various nodes is sent to the second *"/validate"* endpoint, along with the system context.

This information will be included in the second prompt,which can be found in Appendix A.2, which, similarly to the other one, is structured as follows:

1. Again, through the Role-playing technique[24], the model is asked to assume the role of a "cybersecurity expert," specifying its main goal, namely to verify whether that particular threat is applicable to that node.

2. Detailed contextual information relevant to the task is provided, including information about the node, the threat to be analyzed, and the system context. In this way, the model has the necessary information to generate pertinent answers.

3. The task is again subdivided into various sections such as "Understanding the System Graph Relationships," "Node Type Vulnerability Guidelines," and "Target Node Focus and Validation Rules." This gives the model clear instructions on how to approach the assigned task. This structured breakdown of the problem enhances the model's ability to follow step-by-step logical reasoning, aligning with the Chain-of-Thought Prompting approach [25]

4. Through Ontology Definition, the interpretation of the relationships found in the system is made explicit to the model, giving it the information needed to understand the system's structure and behavior, and to evaluate possible indirect attack paths. Ontologies help structure knowledge into hierarchical and non-hierarchical relationships, which can be learned and inferred by Large Language Models [27].

5. Several conditional instructions are given to the model depending on the node type, which provides explanations and examples to guide the model in determining whether the threat actually exists for the node under the analysis or not.

6. The model is forced to generate a specific output format, through Output formatting [26]. A strict JSON format is again requested, and an example is provided, to ensure ease of response processing.

The response requested from the model is as follows:

```
{
      "response": {{
        "Exploitable": <Yes/No>,
        "Reason": <Brief explanation based on the node's
   own characteristics, its security properties, and any
   direct relationships that enable or prevent exploitation>
      }}
}
```

Thus, the task is to determine if the threat is exploitable or not and provide a reasoning element, which was useful for understanding whether the model truly understood the structure of the system, and it is also a useful tool for the system's end user, as it provides information that can be helpful in understanding the origin of the threat.

This process is carried out for all nodes, in order to have a complete analysis of the system. As soon as the process is concluded, a JSON file is created with the results for each node.

## 6.2 Merging of the Results: Graph Reduction

After completing the analysis, two separate results are obtained from the two services in JSON format. The TAMELESS result is saved directly in the cache previously presented; it contains the information on nodes, threats, properties, and relations saved in the database and to which TAMELESS referred.The cache also stores results indicating whether a node can be: compromised, vulnerable, malfunctioning, restored, detected, or fixed.

```
{
...
  "results": {
    "canBeCompromised": [
      {
        "property_name": "canBeCompromised",
        "self": "http://localhost:8080/results/3",
        "entity_id": "1",
        "threat_id": "2"
      }
    ],
   "canBeVulnerable":[]
    ...
  }
}
```

Concerning the results obtained from the analysis with the LLM, after the graph_processor has manipulated the prompt results, a JSON containing all the nodes analyzed is obtained. For each of them,information is provided on the vulnerability that was identified and analyzed, as well as the model's response. Everything is organized in a structure similar to the following:

```
{
"node_1": [
     {
```

```
        "vulnerability": {
             "Tactic": Tactic_name,
             "Technique": Technique_name and MITRE ID ,
        "    Description": Description taken from ATT&CK
         },
         "validation": {
             "Exploitable": Yes or No,
             "Reason": LLM reasoning
             }
         },
    ],
}
```

Once the data is available, the frontend component "GraphResults.jsx", which is responsible for displaying the results to the user, receives this information extracted from the input graph (Figure 6.3).

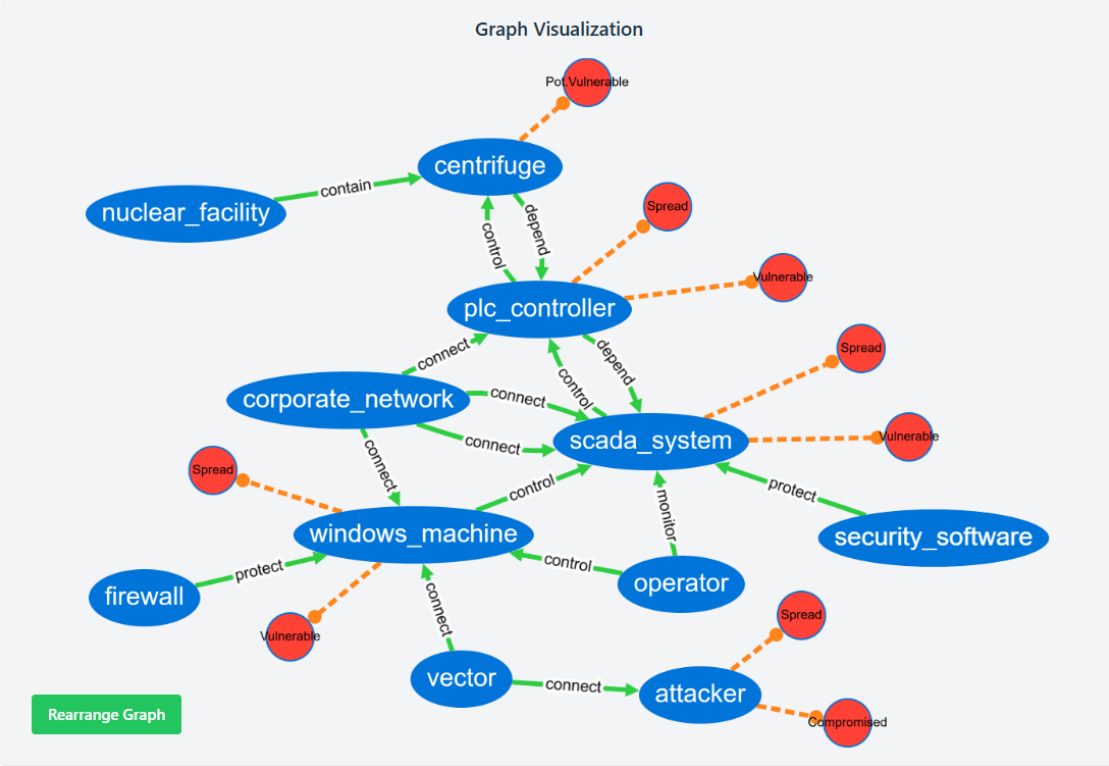


**Figure 6.3:** Stuxnet Input Graph.

The filtering of the results occurs through a series of specific steps that transform the raw data into a more streamlined graphical representation. Initially, a graph is generated starting from the entities and threats present in the TAMELESS cache (Figure 6.4); if the entities appear in the results provided by the analysis, a style is added that visually represents their security status. The edges used to connect the various nodes are always taken from the cache, utilizing the defined relations.This provide a visual representation of the graph with the node states obtained from the formal analysis. If it is decided to display the data together with the results from LLaMA (Figure 6.5), they are modified to obtain a more defined representation. A set of entities is built from the results of TAMELESS and those of the LLM. Subsequently, a new set is created that includes only the entities that appear in at least one of the two sets, by filtering the edges so that they connect exclusively the nodes included in this new set. Moreover, the percentage reduction of the graph is determined by comparing the initial node count with the number of remaining nodes after filtering, providing a quantitative metric of the process's effectiveness. Finally, the number of "added" nodes coming from the LLM analysis is determined, that is, those that were not present in the original TAMELESS list.

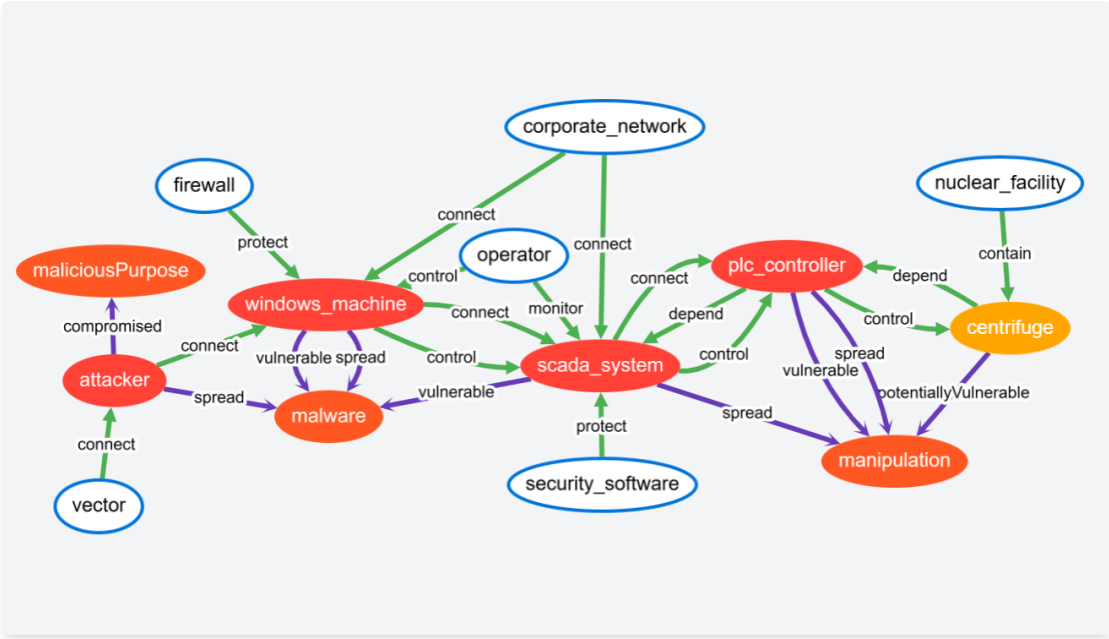


**Figure 6.4:** Stuxnet TAMELESS Results.

The final graph (Figure 6.5) obtained in this way will be a simplified representation of the system, which allows users to concentrate on the critical vulnerabilities and on the connections that actually represent potential attack points. The filtering that is carried out reduces the "noise" by eliminating superfluous nodes and highlighting the essential information, which is particularly useful when analyzing the security of complex systems.

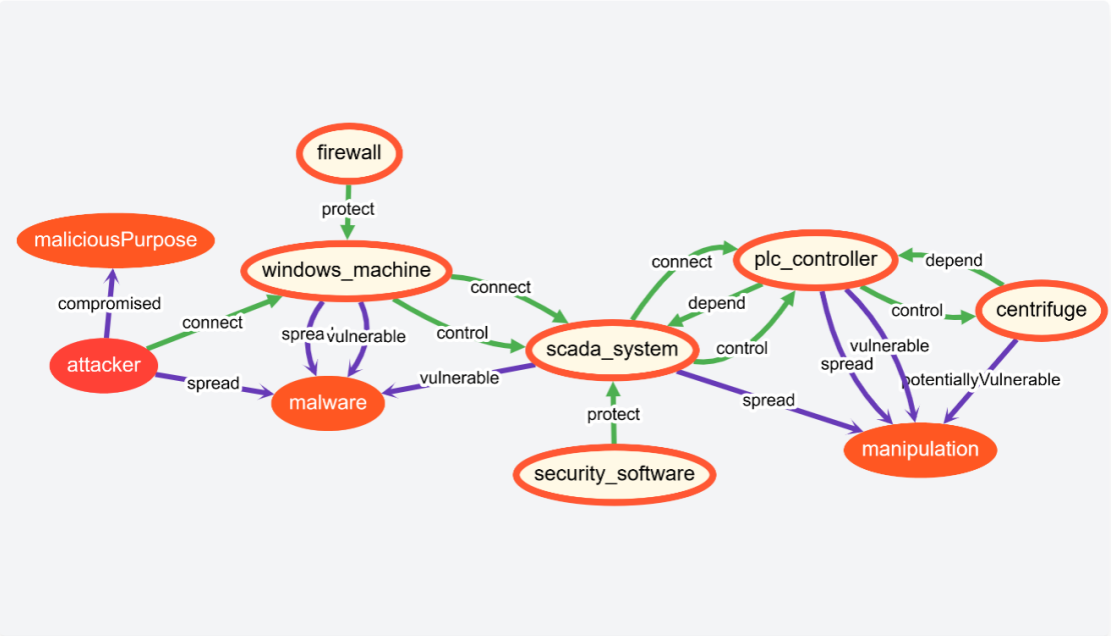


**Figure 6.5:** Stuxnet Combined results.

## 6.3 Calculation of the Probability and the Risk

During the visualization of the results, other useful metrics for the end user are calculated, such as the probability of a node being compromised using only the data provided by TAMELESS and the probability and risk associated with each identified ATT&CK technique.

### 6.3.1 Calculation of the Node's Probability

To define the probability associated with each node, only internal factors of the formal analysis have been taken into account, by determining which elements can

establish the susceptibility of an entity to compromise. It is assumed that if the formal analysis shows that a node is already compromised, then its probability is maximal, i.e., 1.

For all other security states, the probability is calculated by defining the following metrics:

1. **Assessing the state of the node (Vuln)**. If the node is considered vulnerable, it is more likely to be compromised. It is also assessed whether it is malfunctioning, as, even if this does not directly lead to the node's compromise, it might be unable to activate protection mechanisms. Therefore, a numerical value of 1 is assigned if the node is vulnerable, 0.6 if the node is malfunctioning, and 0 if it is in neither state.

2. The number of connected nodes in a compromised state ($Comp_{Conn}$) is calculated. If the node is connected to other compromised nodes (also referring to **Rule 7** of the model), the probability will increase.

3. The number of adjacent nodes that can perform a threat spread ($Threat_{Spread}$) is verified, a factor that significantly contributes to the compromise of the entity.

4. The number of nodes adjacent to the entity under analysis that control it or on which it depends is determined. These nodes, if in a compromised state ($Rel_{Control}$ and $Rel_{Depend}$), are considered factors that may increase the risk of compromise.

5. Finally, the number of nodes that can protect the entity (*Protection*) are verified, not considering a specific threat, but taking into account its protective function. This is regarded as a factor that can lower the probability.

Therefore, the probability will be calulated in this way:

$$P_{\text{compromise}} = w_1 \cdot \text{Vuln} + w_2 \cdot \text{Comp}_{\text{Conn}} + w_3 \cdot \text{Threat}_{\text{spread}} + w_4 \cdot \text{Rel}_{\text{Control}} + w_5 \cdot \text{Rel}_{\text{Depend}} - w_6 \cdot \text{Protection}$$

$$\text{with } \sum_{n=1}^{6} w_n \leq 1.$$

### 6.3.2 Calculation of the Technique's Probability

Once the risk associated with a single node has been calculated, it was decided to define a probability metric for each individual technique identified by the LLM as exploitable. The calculation involves the same factors previously seen but adds information derived from the technique itself. Indeed, MITRE ATT&CK provides indirect indications regarding the complexity an attacker faces in executing an attack. Among the additional factors, is possible to find:

- **Permission**: The permissions required to execute a certain threat were analyzed. The higher the required permissions, such as "Administrator" or "Root", the lower the probability. Conversely if roles like "User" are required, the probability will be higher, because it is easier to obtain such permissions to exploit the vulnerability. The value is extracted from the field *"x_mitre_permissions_required"* of the MITRE technique, and a numerical value was assigned for each: 0.4 if "Root" or "Administrator" privileges are required, 0.6 for "User" privileges, and 0.8 if no specific privileges are required.

- $System_{Req}$: Another factor considered is the system prerequisites necessary for the threat, such as specific configurations, active services, etc. A high number of prerequisites indicates that the technique is more complex, as it requires the victim to have specific characteristics to be vulnerable to the attack. The value is extracted from the field *"x_mitre_permissions_required"* (as stated in the original text). Here too, a numerical value is assigned based on the number of required prerequisites: a value of 0.8 if the list is empty, 0.6 if at most 2 are required, and 0.3 for all others.

- $killChain_{Score}$: The last factor considered indicates the stage of the attack process to which the technique belongs. Each technique can be associated with different attack stages, which are identified by the technique's name. Some stages are considered more complex and difficult to implement than others. Among the various stages, three of the most difficult have been chosen:

  - **Privilege Escalation**: Privilege elevation involve exploiting vulnerabilities or misconfigurations to obtain higher-level access, an action that is difficult to execute given the measures adopted by modern systems.

- **Persistence**: Maintaining access over time is challenging because systems are subject to updates, patches, and continuous monitoring.
- **Defense Evasion**: Evading defenses and masking activities requires highly advanced obfuscation techniques.

If the considered technique belongs to one of these three, it is assigned a value of 0.5; otherwise, it is assigned a value of 0.8.

The final probability is expressed in the following form, combining a topological and systemic evaluation provided by TAMELESS and an operational evaluation provided by MITRE ATT&CK:

$$\begin{aligned} P_{\text{Technique}} = {} & w_1 \cdot \text{Vuln} + w_2 \cdot \text{Comp}_{\text{Conn}} + w_3 \cdot \text{Threat}_{\text{spread}} + w_4 \cdot \text{Rel}_{\text{Control}} \\ & + w_5 \cdot \text{Rel}_{\text{Depend}} - w_6 \cdot \text{Protection} + w_7 \cdot \text{Permission} + w_8 \cdot \text{System}_{\text{Req}} \\ & + w_9 \cdot \text{KillChain}_{\text{Score}} \end{aligned}$$

$$\text{with } \sum_{n=1}^{9} w_n \leq 1.$$

### 6.3.3 Calculation of the Risk

Risk is determined, following the NIST directive, "taking into account the **impact that would result from the events** and the **likelihood of the events occurring**" [28]. It will be associated with each technique.

As the parameter related to the "likelihood of the events occurring", the previously defined $P_{Technique}$ is used; therefore, to obtain the risk, the impact must be calculated.

The impact is determined by taking several factors into account:

1. **The importance of the node within the system structure**: To determine this, the relationships of the node within the system are considered. The number of entities controlled by this node ($RelControl$), the number of entities that depend on it ($RelDepend$), and the number of direct connections with other nodes ($RelConnect$) are calculated. All these factors indicate how

central and important a node is in the system, so its compromise could have greater effects.

2. **Classification of ATT&CK tactics**: Each tactic in the dataset is associated with a tactic based on its purpose, specifically, the phase of the attack in which it can be placed. Since each technique is associated with an intrinsic meaning, its impact can be identified. For example, there are more impactful techniques such as **"Impair Process Control"** which aims to manipulate or sabotage the physical processes of an ICS system, or **"Inhibit Response Function"** which encompasses techniques that aim to disable or hinder security or incident response functions. These tactics, if applied, are very impactful, while there are others such as **"Reconnaissance"** which aim to gather pre-attack information and are less directly impactful on the system. A value has been assigned to each in order to provide a qualitative estimate of the impact.

3. **Identification of defensive elements**: For the analyzed entity, the number of nodes that have a direct relation of *Protection*, *Monitor*, *Check*, and *Replicate* is determined. These are all relationships that can mitigate the impact by reducing the risk that the threat is exploited.

The impact is defined as :

$$\begin{aligned} Impact = w_1 \cdot \mathrm{Rel}_{\mathrm{Depend}} + w_2 \cdot \mathrm{Rel}_{\mathrm{Connect}} + w_3 \cdot \mathrm{Rel}_{\mathrm{Control}} + w_4 \cdot \mathrm{Technique}_{\mathrm{Impact}} \\ + w_5 \cdot \mathrm{Rel}_{\mathrm{Protect}} - w_6 \cdot \mathrm{Rel}_{\mathrm{Check}} + w_7 \cdot \mathrm{Rel}_{\mathrm{Monitor}} + w_8 \cdot \mathrm{Rel}_{\mathrm{Replicate}} \end{aligned}$$

Once the impact calculation is outlined, the defined risk can be defined as :

$$Risk = \mathrm{Impact} \times \mathrm{P}_{\mathrm{Technique}}$$

In this way, the risk would not be delineated between two extremes and thus difficult to understand and compare. For this reason, it was decided to limit it between 1 and 5, and to do this, a *sigmoid transformation* was used, which maps a real input into a limited range, usually between 0 and 1. To obtain extremes between 1 and 5 the following formula was used:

$$\mathrm{Risk} = 1 + 4 \times \left(\frac{x}{x+1}\right) \quad \text{Where} \quad x = \mathrm{Impact} \times \mathrm{P}_{\mathrm{Technique}}.$$

- If $x = 0$ if the product is zero, then

$$\frac{x}{x+1} = 0,$$

  and the risk is

$$\text{risk} = 1.$$

- If $x$ tends to infinity, then

$$\frac{x}{x+1} \to 1,$$

  and therefore the risk tends to

$$1 + 4 = 5.$$

The validation of these formulas was done using case study examples from the research conducted. An example in which the method *Spearphishing Attachment T1566.001* for the entity *Operator* in the scenario **Cyberattack on the Ukrainian Power Grid** reached the value of 2.621 on the previously defined risk scale can be seen in Figure 5.8. This denotes a risk factor that is slightly above the average; thus, it is an important threat vector but not to the extent of being a critical one, as it only gives system access instead of being the main tactic that caused substation sabotage.

# Chapter 7

# Validation

## 7.1 Tests and Results

In this section, the results obtained from the tests performed on the case studies presented in Chapter 3 are shown. First, the context and its components will be introduced, along with the system's input graph. Furthermore, the results from the formal analysis will be presented, detailing the attack paths and any critical issues. Finally, the outcomes achieved with the combination of TAMELESS and LLaMA will be showed, including the reduced graph, followed by an overview of the most relevant techniques identified for each node.

### 7.1.1 Water Supply System Hack

#### Context and Initial Input

As described in Chapter 3, the water supply system hack is an attack that exploits a human vulnerability, in this case, a phishing campaign, that led to the tampering of the component regulating the chemical agents in the water, ultimately poisoning it and making it dangerous for human consumption. This example was reported and tested in the new system; its representation was produced by placing oneself in the role of a Security Architect who attempts to verify the system without having full knowledge of all the possible attack details, but making general assumptions. By taking the system information from the attack report, the resulting graph is as

follows:

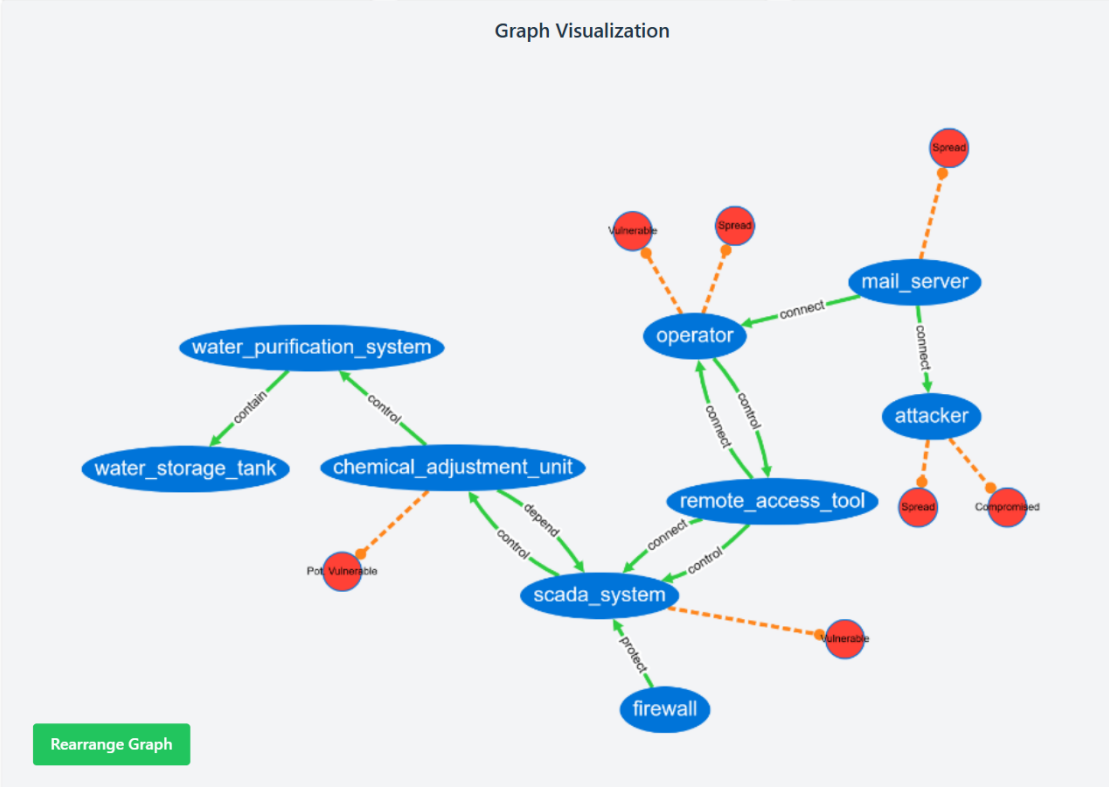


**Figure 7.1:** Water Supply input graph.

The relationships have been determined based on the described structure (Figure 7.1), while the security properties have been defined according to general assumptions, specifically:

- **Attacker (Human)**: **Compromised** by *MaliciousPourpose* and enables the **Spread** of the *PhishingAttack* threat.

- **Operator (Human)**: is assumed to be **Vulnerable** to PhishingAttack given their human nature, and allows the **Spread** of *Manipulation* since they control the SCADA via the Remote Access Tool.

- **SCADA (Physical)**: is found to be **Vulnerable** to *Manipulation* because it is controlled by the Operator.

- **Chemical Adjustment Unit (Physical)**: is **Potentially Vulnerable** to Manipulation, since the SCADA sends commands without real protection.

The Attacker must always be considered as **Compromised** because it is an intrinsic constraint of the model. An entity that is used to compromise another must itself be considered inherently compromised; therefore, the solution is to assume the attacker as compromised, in the sense of its "willingness" to attack the system.

### TAMELESS Results

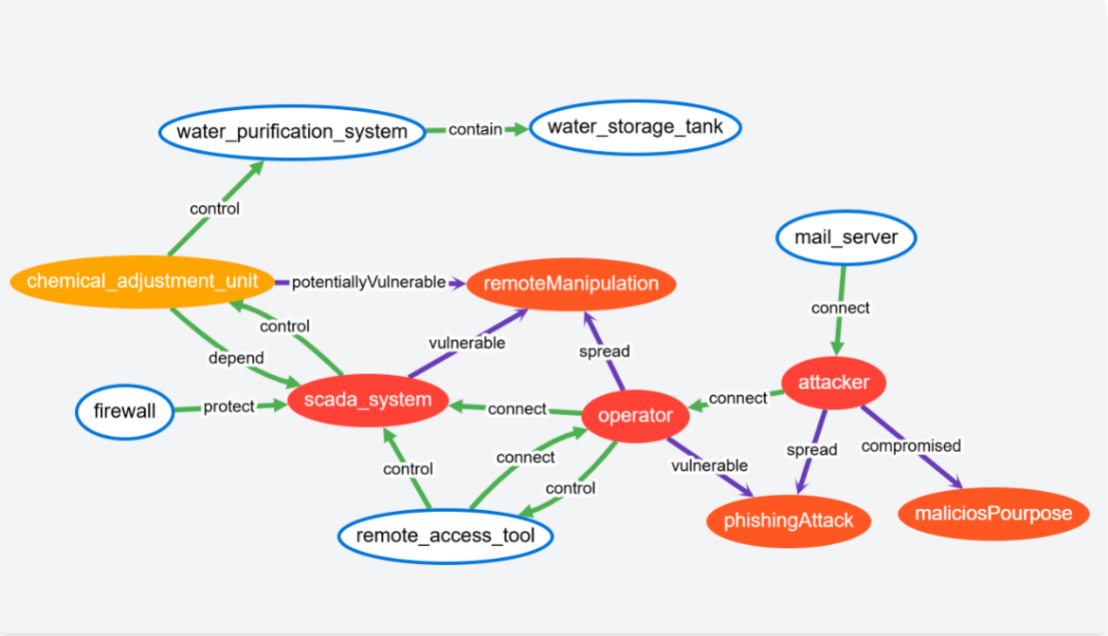


**Figure 7.2:** Water Supply TAMELESS Results.

From the results of the formal analysis (Figure 7.2), it is possible to observe an attack path where the entities *Attacker*, *Operator*, and *Scada System* are **Compromised**. The *Chemical Adjustment Unit* is *Malfunctioning*, indicating that its operation has been compromised, leading to the tampering of the chemical components. Therefore, from this information, an attack path is identified that is compatible with the original one, although no additional information is provided. For this reason, the results of TAMELESS are combined with those obtained from the LLM.

### Combined Results with LLaMA

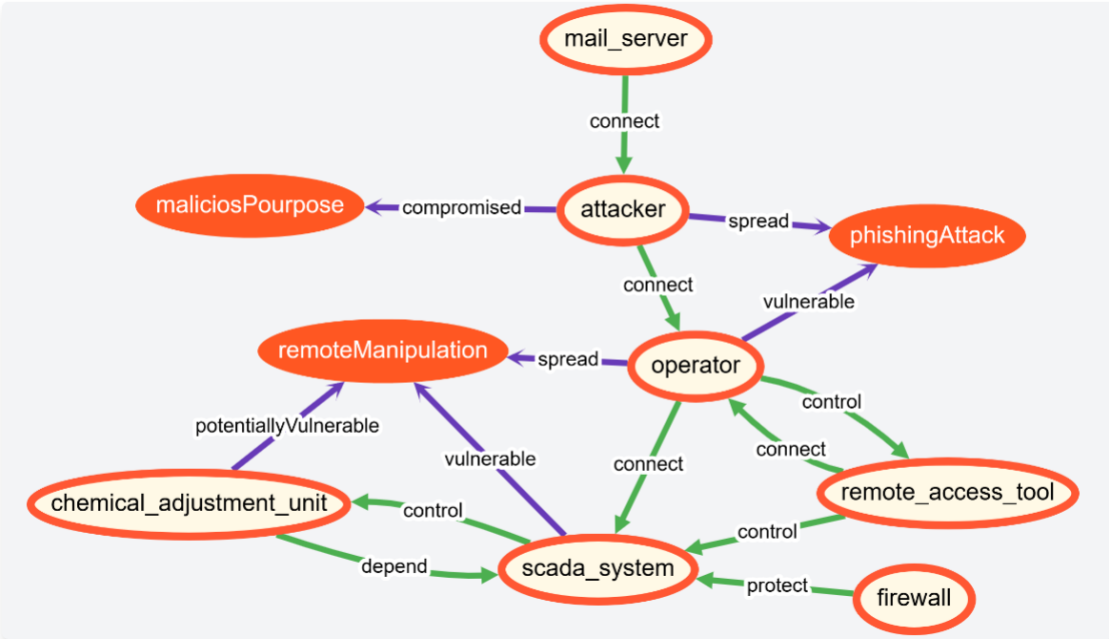


**Figure 7.3:** Water Supply Combined Results.

By displaying and merging the results from the LLaMA analysis (Figure 7.3), it is possible to immediately notice a reduction of the graph, specifically by 22.22%, and the addition of some nodes that were not previously identified by the formal analysis, such as *firewall*, *mail server*, and *remote access tool*. In addition to defining these metrics, the combination of the results aims to provide additional information regarding the compromise and protection of the various entities. Among the most relevant, the following stand out in particular:

- **Attacker** and **Operator**: *Spearphishing Attachment* (T1566.001) and *Spearphishing Link* (T1566.002) are both techniques applicable to human nodes, which presumably represent the method used to carry out the attack. While it is entirely appropriate to highlight these techniques for the Operator, the same is not exactly true for the Attacker, since they do not incur an attack in this context; however, these techniques assigned to the Attacker may be useful if considered as possible ways the attacker could use to threaten the system.

- **Mail Server**: all techniques related to an email server used for phishing campaigns are identified, such as *Spearphishing Attachment and Link*, *Remote Email Collection* (T1114.002), and *Email Collection* (T1114).

- **Remote Access Tool**: both variants of *Spearphishing* (T1566.001 and T1566.002), as well as those related to *Remote Services* (T1210) and *Process Injection* (T1055) are appropriate, because the remote access tool is a software component vulnerable to these specific exploits and effectively acts as a bridge between the Operator and the SCADA system.

- **SCADA System**: the *Remote Service* (T1210) technique is correctly identified, in line with the expected attack vector.

- **Chemical Adjustment Unit**: the techniques highlighted are *Remote Services* (T1210) and *Process Injection* (T1055). The former highlights the remote control by the SCADA, while the latter indicates the injection of possible malicious commands.

### 7.1.2 Flicker Frequency Attack

#### Context and Initial Input

This attack demonstrates how smart system devices, if used with malicious intent and not properly protected, can directly compromise people's safety. In this scenario, the attacker, having access to the LAN in which both the controller and the smart light bulb are present, is able to send commands to the controller through unprotected APIs. This operation allows the modification of the PWD duty cycle, triggering epileptic seizures in predisposed individuals.

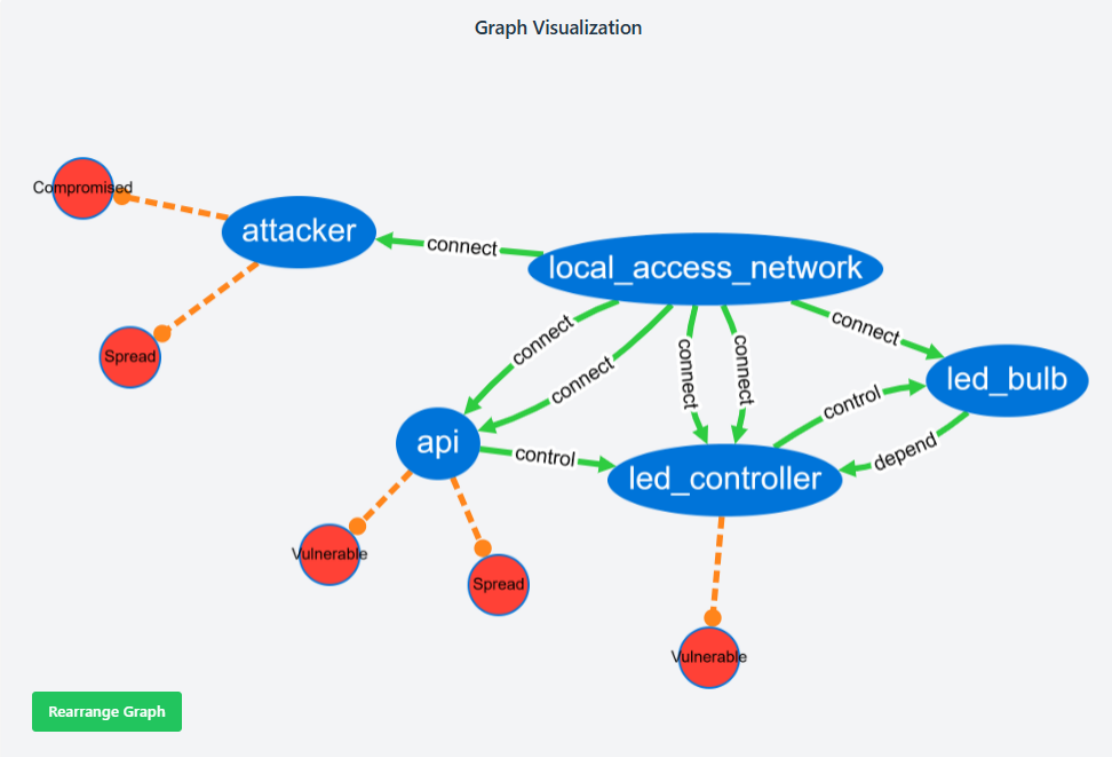


**Figure 7.4:** Flicker Frequency Attack input graph.

The relationships between the nodes were taken from the attack description [17] and are visible in Figure 7.4. Regarding the security properties, the following have been defined:

- **Attacker (Human)**: It is considered **Compromised** due to *MaliciousIntent*, which expresses the desire to attack the system. It is also considered an entity that performs the **Spread** of *UnauthorizedAccess*, since the goal is to access the APIs without authentication.
- **API (Cyber)**: It is classified as **Vulnerable** to *UnauthorizedAccess* because it is not protected by any authentication mechanism and allows the sending of commands, thereby facilitating the **Spread** of *RemoteManipulation*.
- **LED Controller (Cyber)**: It is considered **Vulnerable** to *RemoteManipulation* because it receives commands from the API. Furthermore, it is the component that sends signals to the LED bulb.

- **LED Bulb (Physical)**: This represents the smart light bulb used in the example. It modulates its light based on the commands sent by the controller.

- **LAN (Cyber)**: This represents the internal network to which all elements of the system are connected.

### TAMELESS Results

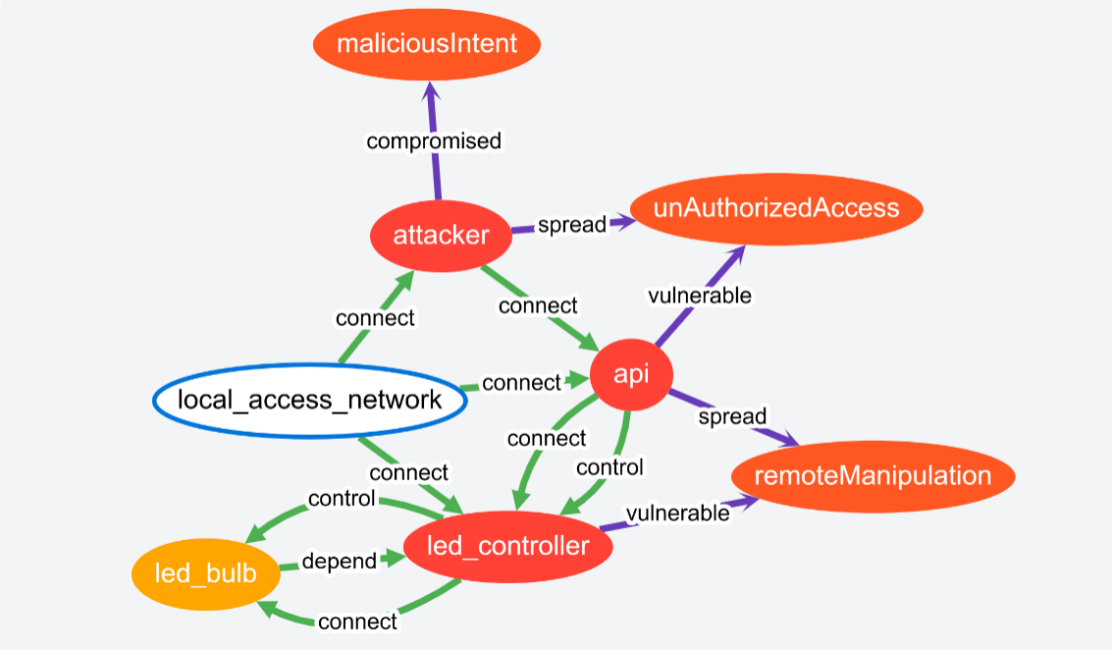


**Figure 7.5:** Flicker Frequency Attack TAMELESS Results.

A well-defined attack path arises from the results of the formal analysis. Specifically, the attacker succeeds in compromising the API, which, by managing the Controller, leads to its compromise. Since the LED bulb depends on an already compromised node, becoming malfunctioning ultimately leads to the success of the attack. Again, the TAMELESS results allow for correct identification of the attacked components and reconstruction of the attack path. However, as in the previous case, detailed information on the exact mechanism that led to the compromise is lacking.

### Combined Results with LLaMA

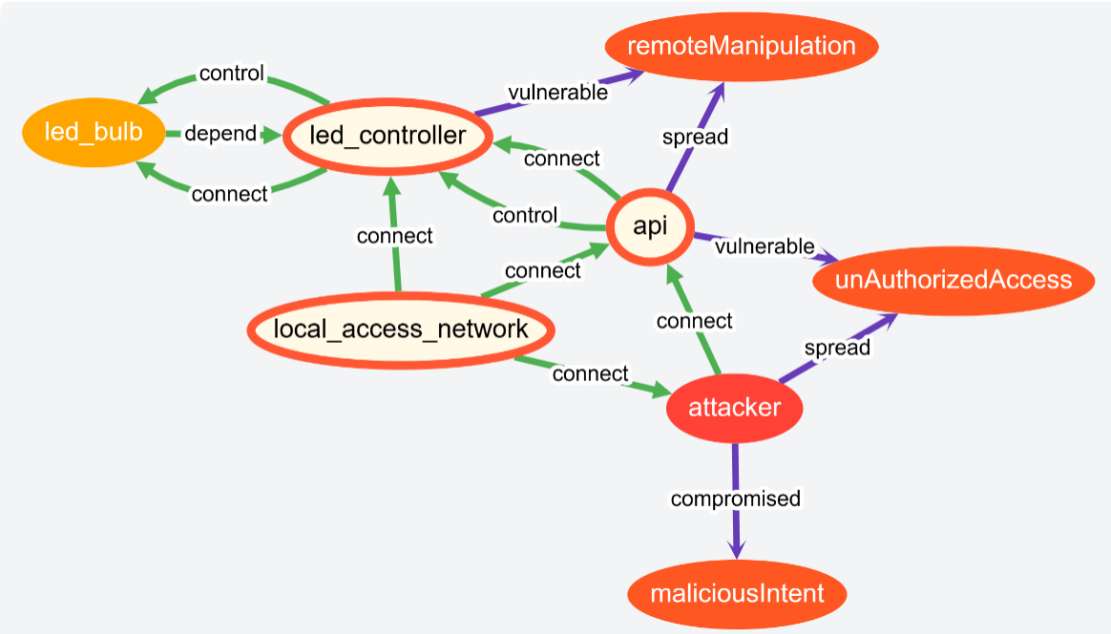


**Figure 7.6:** Flicker Frequency Attack Combined Results.

In this case, the metrics indicate that there is no reduction of the original graph; rather, the only difference is that the node "discarded" by the formal analysis is added to the new graph. This is due to the specificity of the attack, which was developed for a laboratory study. Consequently, the graph is relatively small and represents exclusively the components involved.

Analyzing the techniques identified by LLaMA, it is possible to identify among the most relevant ones:

- **API**: The technique *Exploit Public-Facing Application* (T1190) is identified because the API node is exposed and menages the Led Controller node, representing an entry point for the attack; additionally, *Execution through API* (T0871) is detected, which describes the use of APIs to execute malicious code.

- **LED Controller** and **LAN**: For both nodes, a single applicable technique is identified, namely *Exploit Public-Facing Application T1190*. This technique is

pertinent since both interact with the API node, which is externally exposed and therefore vulnerable.

The identified techniques provide a comprehensive overview of how the attack unfolded; however, given its specificity, the reconstruction is not entirely complete.

### 7.1.3 Cyberattack on the Ukrainian Power Grid

#### Context and Initial Input

The cyberattack on the Ukrainian power grid is an extremely complex attack that unfolded in several phases: (I) the main attack on the substations, which caused a power outage and a complete blackout; (II) the attack on the UPS systems, which prevented the immediate restoration of power; (III) the attack on the call center, which hindered communication with customers, making a timely intervention difficult. To represent this case, a general approach was chosen, without delving into the specific details of the attack, in order to verify whether TAMELESS was capable of identifying the various phases and whether integrating TAMELESS analysis provided a concrete improvement to the overall analysis.

Figure 5.2 shows the initial input graph. The security properties considered are:

- **Attacker (Human)**: It was assumed to be **Compromised** due to *MaliciousIntent*, indicating a willingness to attack the system. It is also considered an entity that performs the **Spread** of a *PhishingAttack* to gain access.
- **Operators (Human)**: They have been classified as **Vulnerable** to *PhishingAttack* due to their human nature and, by interfacing with the SCADA system, they facilitate the **Spread** of *RemoteManipulation*.
- **SCADA System (Physical)**: It is **Vulnerable** to the *RemoteManipulation* threat since it is controlled by the operators.
- **Mail Server (Cyber)**: Although it does not have dynamic properties assigned, it plays a role in connecting the attacker to the system.
- **VPN Access (Cyber)**: This cyber node represents the access point through a virtual private network, which connects the operator to the SCADA system.

- **Uninterruptible Power Supply (Physical)**: This is a critical physical component for ensuring operational continuity in the event of a power outage.
- **Substation (Physical)**: This represents the transformation station, a key nodal point in the energy distribution network.
- **Power Grid (Physical)**: This physical node represents the entire electrical infrastructure, acting as a link between the substation and the end users.
- **Telephone Line (Physical)**: This entity is used to connect the call center to the attacker and to the residential users.
- **Call Center (Physical)** and **Call Center System (Cyber)**: The physical node **Call Center** and its corresponding cyber counterpart, **Call Center System**, represent the communication and support system.

This comprehensive representation, which integrates both the nodes with specific security properties and supporting nodes, offers a detailed view of the system's structure and the possible attack vectors, providing a valid starting point for the overall security analysis.

### TAMELESS Results

Figure 5.6 shows the results of the formal analysis conducted with TAMELESS. The graph indicates that the nodes **Attacker**, **Operators**, and **SCADA System** are in a compromised security state, while the **Substation** is malfunctioning, confirming the success of the attack. The main attack path, which starts at the attacker and ends at the substation, is correctly identified; however, without further specific information, the other attacks cannot be fully detected.

### Combined Results with LLaMA

With the filtering performed by incorporating the results obtained from LLaMA analysis (Figure 5.7), the attack path becomes clearer and simplified. In particular, a total graph reduction of 33.33% is observed and four nodes have been added compared to the formal analysis, namely **Call Center System**, **Mail Server**, **Uninterruptible Power Supply**, and **VPN Access**. The nodes related to

parallel attacks are thus identified with greater precision, providing further guidance to the security architect during system analysis.

Observing the identified techniques, the following emerge:

- **Mail Server**: Techniques such as *Spearphishing Attachment* (T1566.001) and *Spearphishing Link* (T1566.002) are identified, reflecting the initial attack vector; moreover, execution techniques such as *Command and Scripting Interpreter* (T1059.001) and *Process Injection* (T1055) appear plausible for executing malicious code.

- **Operators**: *Spearphishing* techniques (both attachment and link) are detected, consistent with the human nature of the operators.

- **VPN Access**: The technique *Exploit Public-Facing Application* (T1190) appears applicable, indicating a potential weakness at the entry point.

- **SCADA System**: The model identifies two techniques, *Command and Scripting Interpreter* (T1059.001) and *Process Injection* (T1055), which are consistent with the goal of compromising the SCADA system (for instance, to install the BlackEnergy malware and facilitate lateral movement within the network).

- **Uninterruptible Power Supply**: The technique *Loss of Protection* (T0837) is reported, deemed appropriate considering the tampering experienced by the UPS systems.

- **Call Center System**: Techniques such as *Spearphishing Attachment* (T1566.001), *Spearphishing Link* (T1566.002), *Phishing* (T1566), and *Remote Services* (T1210) are identified; although plausible, they are not entirely consistent with the actual dynamics of the attack.

The addition of this information provides a more complete view of the attack dynamics: although the techniques are fairly general, they offer a plausible insight into the threats that can compromise the security state of the various entities.

### 7.1.4 Cyberattack on the Ukrainian Power Grid - Detailed

#### Context and Initial Input

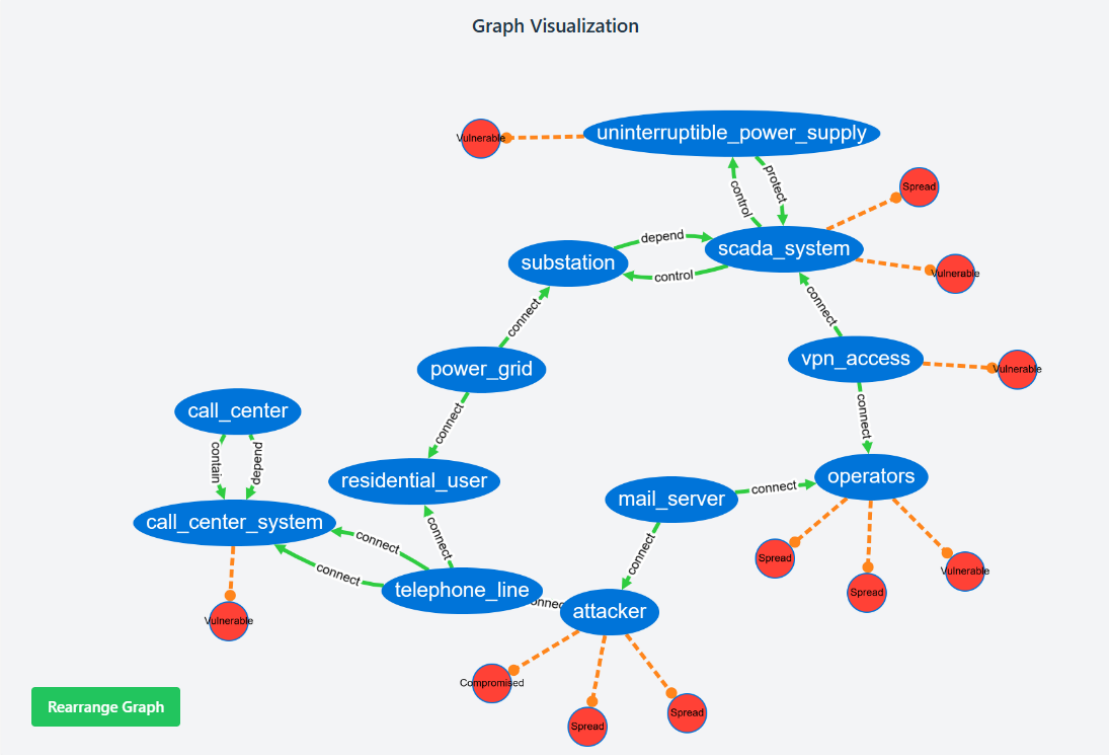


**Figure 7.7:** Ukrainian Power Grid - Detailed input graph.

Based on previous results, further information was incorporated into the analysis to achieve a deeper understanding of the attack. The enhanced input graph (Figure 7.7) includes additional security properties that more accurately represent real-world attack scenarios.

The objective is to verify whether the augmented system can recognize previously occurred attacks while providing a detailed reconstruction of the attack path for the security architect. This enriched input simulates not only direct compromises, but also secondary effects that cascade through interconnected nodes, thereby providing a more comprehensive view of the potential vulnerabilities within the network.

Compared to the previous example, the following security properties have been

added:

- **Attacker (Human)**: it is explicitly indicated that the attacker performs the **Spread** of the *DenialOfService* threat.
- **Operators(Human)**: the property **Spread** of *CredentialTheft* has been added. This means that the attackers could compromise operators and steal VPN credentials to access the network.
- **VPN Access**: since the access credentials have been leaked, the node is now considered **Vulnerable** to *CredentialTheft*.
- **SCADA System (Physical)**: in addition to being **Vulnerable** to *RemoteManipulation*, it also allows for **Spread** of that specific threat, since the tampering of the SCADA has led the compromise of the UPS.
- **Uninterruptible Power Supply (Physical)**: this node is classified as **Vulnerable** to *RemoteManipulation* due to potential tampering, which could lead to a shutdown of the power backup systems.
- **Call Center System (Cyber)**: explicitly represented as **Vulnerable** to *DenialOfService*.

**TAMELESS Results**

From the results of the formal verification (Figure 7.8), it emerges that the entities **Attacker**, **Operators**, **Scada System**, **Call Center System**, and **Uninterruptible Power Supply** are all classified as *Compromised*. While, the nodes **Call Center** and **Substation** are in a state of *Malfunctioning*, indicating operational disruptions. Finally, the **VPN Access** node is confirmed as **Vulnerable**, which aligns with the input assumptions regarding leaked credentials.

The main attack path, which starts from **Attacker** and ending at **Substation**, is correctly identified, demonstrating the model's capability to trace a direct, high-impact breach. Futhermore, secondary attack paths that affect the **Call Center System** and **Uninterruptible Power Supply** are also successfully traced, highlighting the cascading effect of compromised nodes throughout the network.

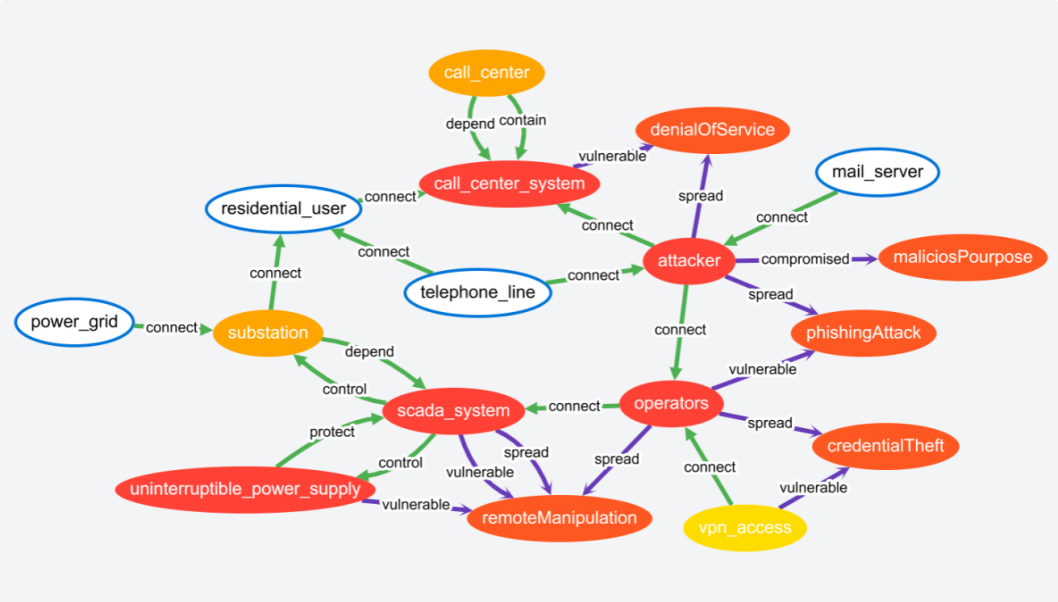


**Figure 7.8:** Ukrainian Power Grid - Detailed TAMELESS Results.

### Combined Results with LLaMA

The combined results (Figure 7.9), obtained by integrating the analysis performed with the LLM (LLaMA), show an 8.33% reduction of the graph and the addition of three nodes compared to the formal analysis, those related to **Mail Server**, **Power Grid** and **Telephone Line**. In particular, the additional techniques detected are:

- **Attacker**: techniques such as *Application Exhaustion Flood* (T1499.002), *Spearphishing Attachment* (T1193) and *Masquerading* (T1036.001) that highlight attack methods compatible with the real case.
- **Call center**: for the call center node, treated as physical, the technique *System Service Discovery* (T1007) has been identified. This technique allows the attacker to detect active services and map the internal network, providing essential information to plan further phases of the attack.
- **Call Center System**: techniques such as *Service Stop* (T0881) and *Denial of Service* (T0837) highlight the operational impact of an attack on the call center system. The interruption of critical services can compromise the ability to respond and coordinate, exacerbating the consequences in case of an attack.

- **Operators**: techniques such as *Phishing* (T1566) and *Spearphishing Attachment* (T1193) have been identified by the model as being used to compromise access to control systems.

- **VPN Access**: the technique *Exploit Public-Facing Application* (T1190) demonstrates a vulnerability in public-served port expositions that are accompanied by the software. This type of security breach could offer a potential intruder the opportunity to break into the network through poorly protected interfaces, thus making it possible to move laterally within the infrastructure.

- **SCADA System**: in addition to the techniques already identified, the technique *Exploitation of Remote Services* (T1210) emerges, which allows for remote access to SCADA systems.

- **Uninterruptible Power Supply**: the techniques *Device Restart/Shutdown* (T0816) and *Denial of Service* (T1499) are added to indicate the shutdown of the UPS.

- **Power Grid**: the technique *Loss of Control* (T0827) is recognized, namely, it is a situation in which operators are not able to have control over the grid, and can cause prolonged outages and potential physical damage to facilities.

The additional informations provided as input lead to a significant improvement in the final result compared to the previous case, providing a complete view of the attack and demonstrating that the integrated system with LLaMA analysis allows the operator to reconstruct in detail an attack that was suffered. Furthermore, by following these indications for mitigation and detection, the integrated system not only allows the reconstruction of the entire attack path, but also provides practical guidance to prevent similar incidents from recurring.

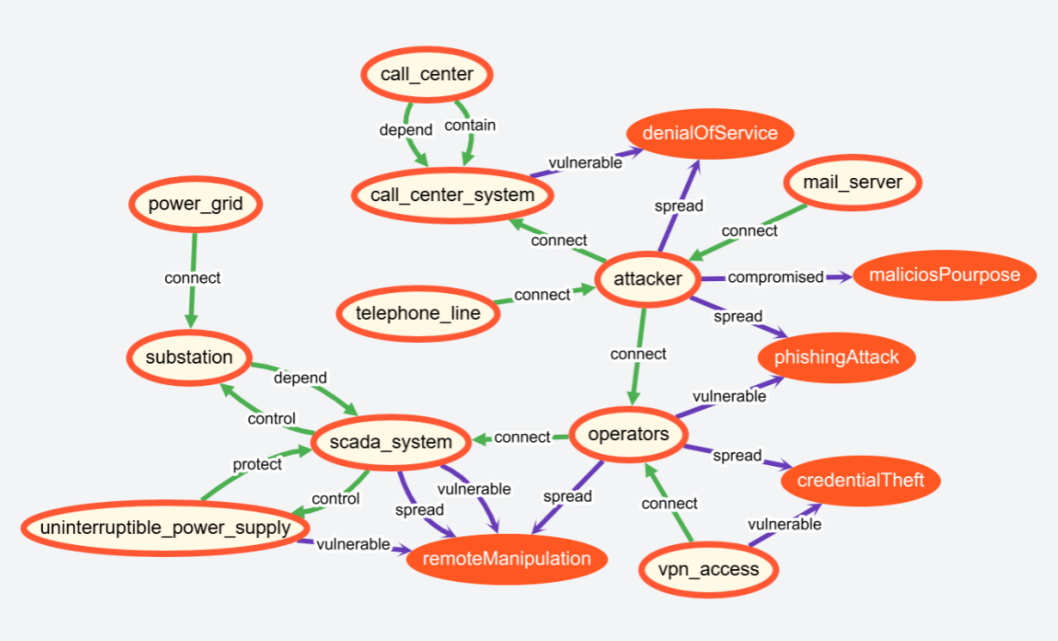


**Figure 7.9:** Ukrainian Power Grid - Detailed Combined Results.

### 7.1.5 Stuxnet

#### Context and Initial Input

In the Stuxnet case study, the attack targeted the Natanz nuclear facility in Iran in 2010, using extremely sophisticated zero-day exploits designed to sabotage industrial control systems. The input graph, as shown in Figure 6.3, contains the basic information needed to generally describe a potential attack.

In the model, the key entities are:

- **Attacker (Human)**: a node which, by assumption, is *Compromised* and possesses the ability to **Spread** the *Malware* threat. It is preset as the initially compromised element to initiate the attack path.
- **Vector (Physical)**: it represents the physical channel used to introduce the malware into the network. No significant dynamic properties have been assigned, as its role is solely that of a vehicle.
- **Windows Machine (Cyber)**: a node that exhibits properties such as being

**Vulnerable** to a *Malware* threat and having the ability to **Spread** it. This reflects how the malware was introduced into the system.

- **Corporate Network (Cyber)**: a node that connects the IT entities and facilitates the lateral movement of the attack, despite not presenting specific security properties.
- **SCADA System (Physical)**: a node that controls industrial processes; it has been labeled as **Vulnerable** to *Malware* and is capable of **Spread** via *Manipulation* actions, given its ability to control the PLC Controller.
- **PLC Controller (Physical)**: this node is marked as **Vulnerable** and has the ability to **Spread** threats related to *Manipulation*.
- **Centrifuge (Physical)**: it represents the centrifuges used for nuclear enrichment; it is marked as **Potentially Vulnerable** to *Manipulation*, highlighting the uncertainty regarding the degree of exposure while indicating the potential physical impact of the attack.
- **Firewall** and **Security Software**: both support nodes, of **Cyber** type, representing the physical infrastructure, network defense systems, and cybersecurity solutions. Although they do not display dynamic input properties, they are connected via relationships (e.g., *Protect*) that contextualize their defensive role.

In summary, the input graph for Stuxnet provides a comprehensive representation of the attack path, with the goal of not getting too specific but providing general indications of a possible attack, in order to assess the potential of the techniques identified with LLaMA.

### TAMELESS Results

The formal analysis conducted for the Stuxnet case by TAMELESS, as shown in Figure 6.4, highlights a well-defined attack path: the entities **Attacker**, **Windows Machine**, **SCADA System**, and **PLC Controller** are all *Compromised*, while the entity centrifuge is in a critical state of *Malfunctioning*.

In particular:

- The **Attacker** node is confirmed as compromised, serving as the origin of the malware.
- The **SCADA System** and the **PLC Controller** show compromise that propagates through control and dependency relationships, representing the spread of the malware toward the industrial systems.
- The **Centrifuge** node is particularly critical, being the final element of the attack: its vulnerability, indicated as *PotentiallyVulnerable*, underlines the risk of physical damage and the inevitable compromise following the manipulation of industrial controls.

The TAMELESS model, by applying its derivation rules, confirms the logical flow of the attack: the initial compromise of the **Attacker** node propagates through the system connections, resulting in a cascade of compromises up to the centrifuges. The support nodes, such as **Firewall** and **Security Software**, although not exhibiting initial vulnerability dynamics, do not intervene in countering the propagation of the attack.

### Combined Results with LLaMA

The integration of the results obtained from the formal analysis of TAMELESS with those derived from the LLM (Figure 6.5) has led to a more detailed representation of the attack path. The combination of the two approaches resulted in a reduction of the overall graph by 36.36% and the addition of supplementary nodes, such as those related to protection purposes, including the **Firewall** and the **Security Software**.

The combined results show:

- **Windows machine**: The node is particularly vulnerable, and LLaMA has identified the following relevant techniques: *CMSTP T1191*, which exploits the CMSTP tool to install Connection Manager profiles, facilitating infiltration; *Netsh Helper DLL T1546.007*, which allows persistence to be established by executing malicious content. Techniques related to the cyber nature of the node are also identified, such as *Spearphishing Attachment T1566.001* and *Spearphishing Link T1566.002*.

- **SCADA System**: For this node, techniques have been identified that highlight its exposure, such as: *Command and Scripting Interpreter T1059.001* that enables the execution of malicious commands via scripting interpreters; *CMSTP T1191* and *CMSTP T1218.003* that both show how the system can be compromised through the delivery of malware; *Netsh Helper DLL T1546.007* that reinforces the risk of persistence through the execution of malicious DLLs.
- **PLC Controller**: The entity is vulnerable to techniques that compromise its monitoring capability; for example, *Detect Operating Mode T0868* and *Detect Program State T0870* allow the collection of information on the current state of the controller, facilitating targeted attacks. Additionally, control techniques such as *Program Upload T0845* enable the loading of programs from the PLC, extracting critical information about the industrial process.
- **Centrifuge**: As the final target of the attack, this node has been identified with techniques of great importance: *Compromise Software Dependencies and Development Tools T1474.001* indicates how the manipulation of software dependencies can compromise the node, even when applied in a physical context; *Exploitation of Remote Services T0866* highlights the risk of lateral movement through the exposure of remote services.
- **Firewall**: Techniques have been identified for discovery that are potentially non-damaging but which highlight possible issues in the security configuration, such as *Security Software Discovery T1518.001*, *Internet Connection Discovery T1016.001*, and *Network Service Scanning T0841*.
- **Security Software**: Finally, this node shows vulnerabilities that can be exploited to compromise the system's defenses, such as *Security Software Discovery T1518.001* and *Internet Connection Discovery T1016.001*, which indicate that even security systems can be subject to discovery techniques. Additionally, *System Service Discovery T1007* enables the detection of active services, providing useful information to the attacker.

The combination of the two sets of results provides the security architect with a fairly complete and integrated view of the Stuxnet attack path, with greater granularity in identifying the attack techniques applicable to each node by drawing

on insights from the MITRE ATT&CK dataset, thus offering practical guidance for the detection, mitigation, and prevention of future attacks.

### 7.1.6 Stuxnet - Detailed

#### Context and Initial Input

Just as was done for the case of the "Cyber Attack on the Ukrainian Power Grid", it was decided to test the case study again, providing the input graph with more specific information. The objective is to verify the behavior of the system integrated with LLaMA and to demonstrate that the model can indeed help in finding useful and detailed information regarding the threats that can compromise the system.

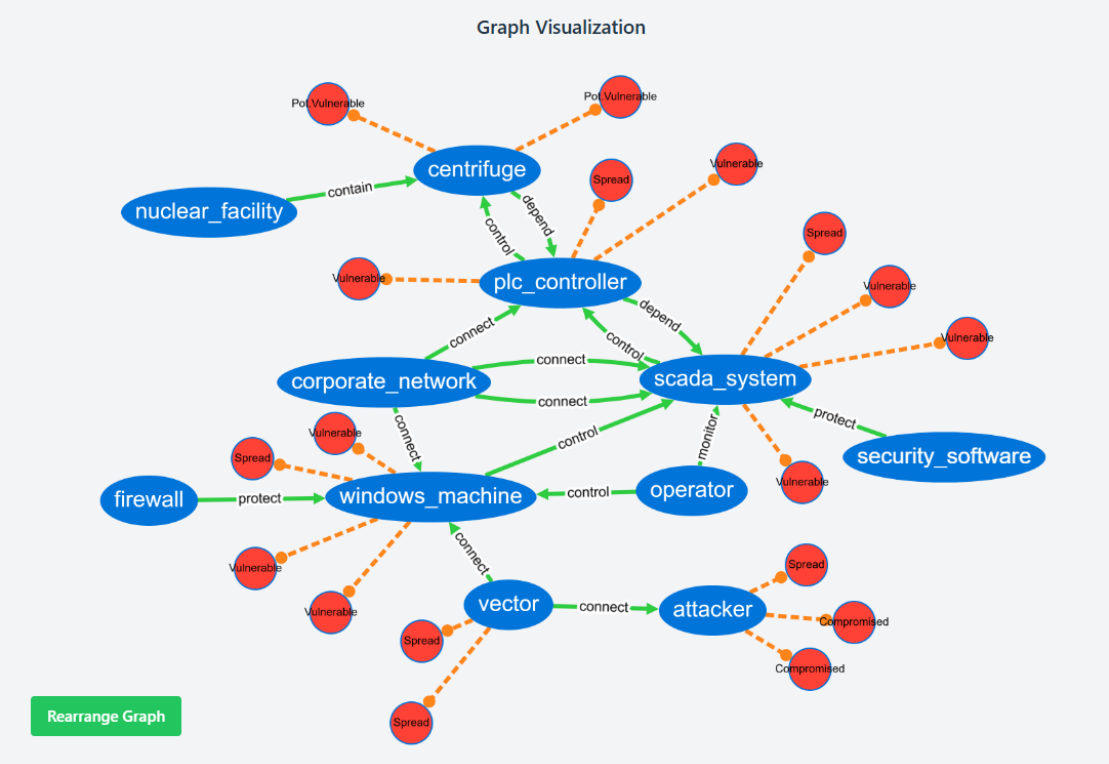


**Figure 7.10:** Stuxnet - Detailed Input Graph.

Compared to the previous test, the entities have remained unchanged, while the security properties have been modified:

- **Attacker (Human)**: The node has been recognized as capable of performing the **Spread** of the *Stuxnet Payload*, thereby more specifically highlighting its initial role in the attack.

- **Vector (Physical)**: In the previous case, no properties were assigned. In this new test, it is instead considered capable of **Spread** of the *Malware* and, specifically, of the *Stuxnet Payload*, recognizing its actual role as the initial attack vector.

- **Windows Machine (Cyber)**: The node has maintained the properties previously identified but has added two more. It has been recognized as **Vulnerable** to *Zero day Exploit* and *LNK*, both vulnerabilities identified during the real attack.

- **SCADA System (Physical)**: In addition to the previous properties, it has been labeled as **Vulnerable** to *Manipulation* and *SiemensPLCExploit* with the ability to **Spread** through actions of *Manipulation*, based on the information obtained from the attack.

- **PLC Controller (Physical)**: For this node, its vulnerability to the malware has been explicitly stated, and it has been named *StuxnetController*.

- **Centrifuge (Physical)**: The node has been marked as *potentiallyVulnerable* not only to manipulation but also to *centrifugeOverload*.

The input data for this test are much more detailed and specific than the previous case, thus, it provides a more accurate attack path representation. In this way, it is feasible to run tests to observe how the model behaves and to check whether it is reliable.

### TAMELESS Results

The result obtained from TAMELESS for this new test remained unchanged compared to the previous case. The previously defined attack path is confirmed: the entities **Attacker**, **Windows Machine**, **SCADA System**, and **PLC Controller**

are all *Compromised* while the entity **Centrifuge** is in a critical state of *Malfunctioning*. Naturally, the graph contains specific information on the threats that were previously identified, the same ones that were specified in the input.

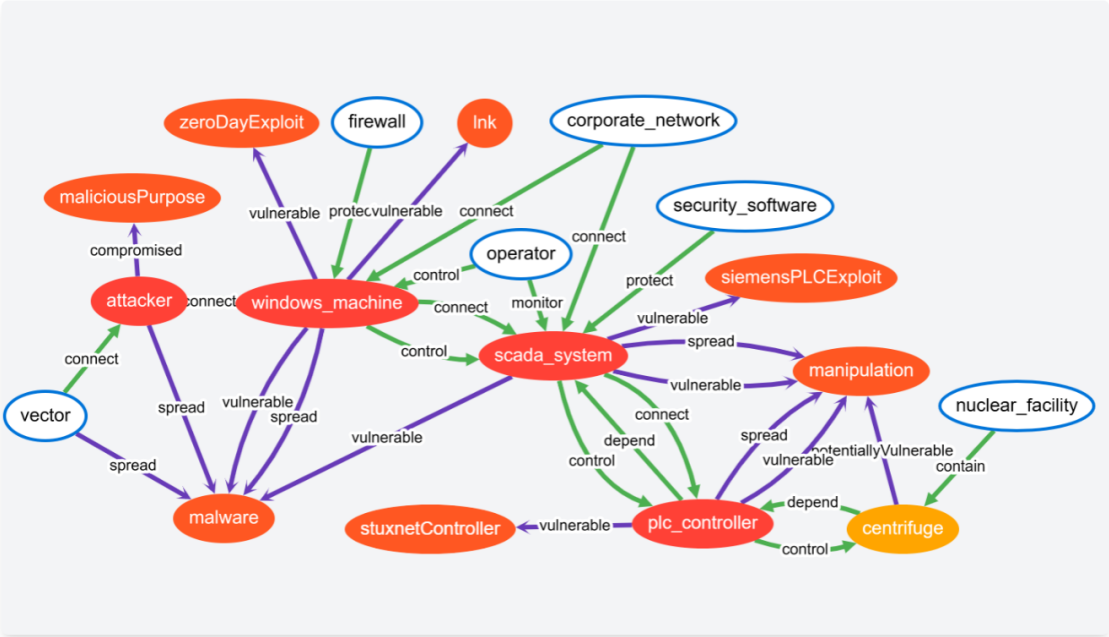


**Figure 7.11:** Stuxnet - Detailed TAMELESS Results.

### Combined Results with LLaMA

The results obtained using the LLaMA module (Figure 7.12) differ significantly from those of the previous case. In this test, a 9.09% reduction of the input graph is observed. This value is lower than in the previous case, but as noted in the "Ukrainian Power Grid - Detailed" example, it occurs because with more information the model becomes more precise and includes more nodes. Listed below are all the techniques deemed most relevant for the case study:

- **Attacker**: Compared to the previous case, possible techniques applicable to this node have been identified, although they remain completely generic regarding its human nature. The identified techniques are: *Spearphishing Attachment* (T0865), *Spearphishing Link* (T1566.002), and social engineering techniques that can be used as an initial access point to the system.

- **Vector**: Also in this case, techniques that were not identified in the previous one have been recognized. Techniques such as *Code Injection* (T1055.001) and *Boot Record Infection* (T1542.001) perfectly represent the role of the vector as the initiator of the attack.

- **Windows machine**: For this node, LLaMA has identified, in addition to the *CMSTP T1191* technique, other relevant techniques related to the context of the attack. For example: *Remote Services* (T1021) used for lateral movement; *Process Injection* (T1055.001) since code injection is a key technique for malware propagation; and *Abuse of Service* (T1068) used by Stuxnet to obtain elevated privileges.

- **SCADA System**: LLaMA has identified several techniques related to the context of the attack. For instance, *Engineering Workstation Compromise* (T0818) highlights how the compromise of workstations can represent an entry point for gaining control of the system. *Program Organization Units* (T0844) employs the code-prepending technique to infect the PLCs, thereby compromising the control logic. *Loss of Control* (T0827) indicates the achievement of a critical state in which the industrial system loses its ability to operate safely; Finally, *Remote Services* (T1021) underscores how the connection between the **Windows Machine** node and the **SCADA System** facilitates the lateral movement of the attack.

- **PLC Controller**: The analysis conducted by LLaMA highlights some techniques that the attack will need to use for its success. In particular, the *Standard Application Layer Protocol* (T0869) technique demonstrates how the connection to the vulnerable SCADA system makes the node susceptible to protocol level exploits. The application of *Detect Operating Mode* (T0868) underlines the importance of monitoring the operational state.

- **Centrifuge**: In this particular node, the analysis has identified several techniques that reflect the critical role of the target. For example, *Loss of Safety* (T0880) highlights how a compromise of this node may lead to dangerous operating conditions, while *Masquerading* (T1036) indicates the possibility of masking anomalies through the manipulation of operational parameters.

- **Firewall**: LLaMA approach highlights various relevant techniques. For example, *Connection Proxy* (T0884) suggests that weak configurations may expose the node, while*System Network Configuration Discovery* (T1016) leverages the firewall's role as a network information hub.

- **Security Software**: Finally, the analysis highlights techniques useful for compromising defenses. In particular, *Server Software Component* (T1505) shows how the abuse of extensible functionalities of server components can compromise the node's security, while *Security Software Discovery* (T1518.001) confirms that an inadequate configuration makes the system susceptible to information gathering techniques.

From the obtained techniques, it can be seen how the model succeeds in currently representing the threats that may afflict the system, providing the necessary tools for its understanding and verification.

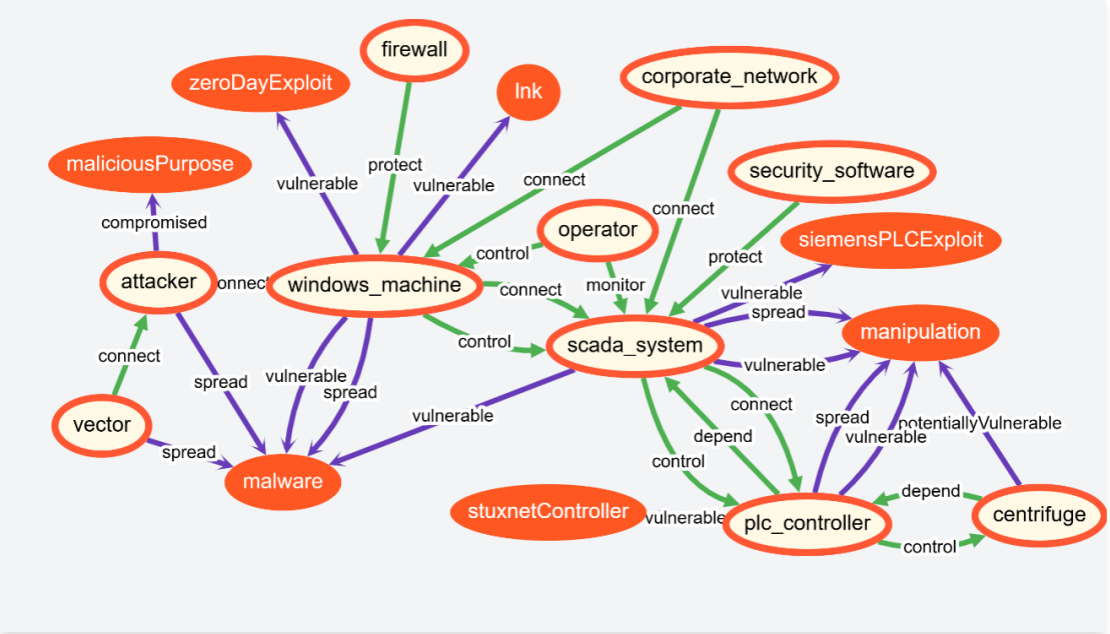


**Figure 7.12:** Stuxnet - Detailed Combined Results.

## 7.2 Comparative Analysis and Discussion

This section present a comparison between the results of the purely formal analysis performed with TAMELESS and those obtained through the integration of LLaMA with MITRE ATT&CK. The objective is to understand how the new module may affect the final threat analysis, both qualitatively and quantitatively.

### 7.2.1 Quantitative Analysis of Metrics

#### Graph Reduction

A useful metric introduced with the new system is the reduction of the attack graph. It indicates how effectively the filtering reduces potential nodes that add complexity to the graph, thus making it harder to identify the attack path.

Figure 7.13 summarizes the reductions in the different scenarios:

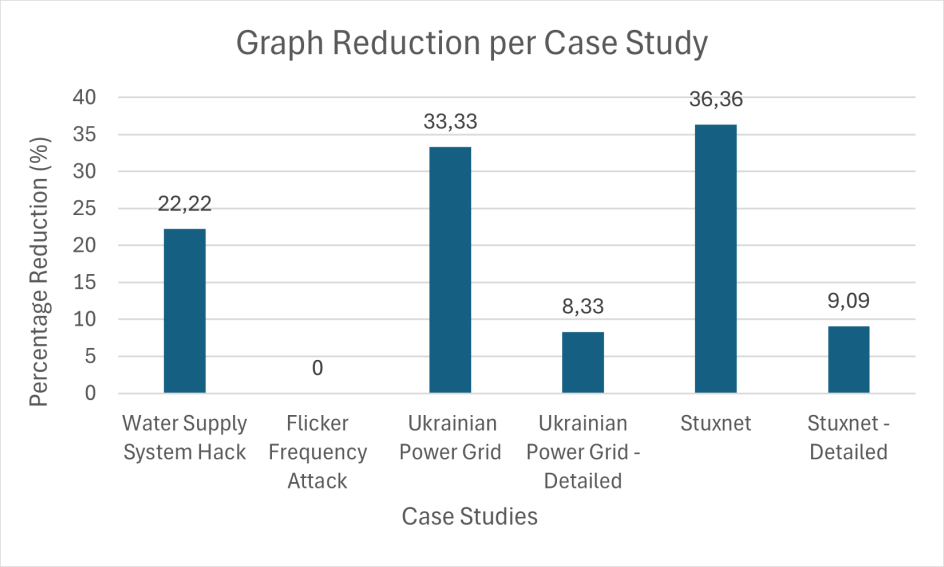


**Figure 7.13:** Graph Reduction per Case Study.

From these values, several considerations emerge:

1. In complex cases such as "Cyber Attack on the Ukrainian Power Grid" and "Stuxnet", which are multi-step and sophisticated attacks, a substantial reduction of the graph is observed. The integrated analysis is particularly useful in

very complex contexts, where the information extracted from LLaMA's results helps filter out less relevant nodes.

2. For scenarios like "Water Supply System Hack" there is a moderate reduction of the final graph, while "Cyber Attack on the Ukrainian Power Grid - Detalied" and "Stuxnet - Detalied" show an even lower percentage of reduction. In these cases, the initial graphs were already more detailed, leading the LLM module to achieve a smaller reduction in nodes, even while introducing more specific information.

3. Finally, in the "Flicker Frequency Attack" the graph remains unchanged. The scenario was already focused on the attack, leaving little space for further filtering.

Table 7.1 reports the graph reduction statistics:

**Table 7.1:** Statistical Summary of Graph Reduction Percentages

| **Statistic** | **Value (%)** |
|---|---|
| Average (Mean) | 18.22 |
| Median | 15.66 |
| Maximum | 36.36 |
| Minimum | 0.00 |

These data indicate that, on average, the new system leads to a significant simplification of the graph, thereby improving the understanding of the attack paths.

### Probability and Risk Metrics

Another metric that has been added concerns the probability and risk that are primarily associated with each node and the corrisponding techniques identified. From the data provided in the tests, it is noticeable that a risk, which integrates probability with impact, is determined by the importance of the node within the system's structure and by the identified MITRE technique. High risk values are associated with nodes that control critical and fundamental processes. For example:

- The SCADA System in the case of "Water Supply System Hack" has an associated risk of 3.128 out of 5 for the 'Remote Services' technique, as it is a central node that directly controls the Chemical Adjustment Unit.

- The APIs, with regard to the "Flicker Frequency Attack", being an important node that allows control of the 'Led Controller', have an associated risk of 3.068 out of 5.

- Also, the SCADA System for the case study "Cyberattack on the Ukrainian Power Grid" has an associated risk value of 3.564 for the *Process Injection* technique and 3.28 for the *Exploitation of Remote Services* technique in the "Verified" case study.

- Finally, for "Stuxnet", an example is always the SCADA system, with a value of 3.629 for the *Netsh Helper DLL* technique and a value of 3.221 for the *Remote Services* technique in the "Detailed" test.

All other nodes have a slightly lower risk, which aligns with the expected results.

Regarding probability, it is noticeable that high values are often associated with nodes that are directly exposed or have dependency relationships with compromised nodes. Moreover, by definition, all nodes identified as compromised by the system have a probability equal to 1. This probability slightly decreases based on the type of technique associated with the node, as it depends on the required permissions, the necessary system requirements, and the type of tactic to which the identified technique belongs. For all other nodes, the probability ranges between medium and high when the node has an altered security state. For example, in the case of "Stuxnet", the centrifuge has a value of 0.77. Similary, if a node is highly dependent on compromised nodes, such as the Corporate Network, which has a probability of 0.70, its probability is also elevated.

Other nodes that do not have particular relationships with compromised nodes or altered security states have a lower probability, as with Vector, again for the "Stuxnet" case study. It has a probability of being compromised of 0.20, which rising to 0.3 for some associated techniques.

These values enable the end user to focus on the most critical nodes of the system, highlighting their importance within the structure, so that effective countermeasures can be identified to prevent a potential attack.

### 7.2.2 Qualitative Discussion of Results

#### Addition of Nodes

Another metric introduced by the new module is the addition of nodes that were not included in the formal analysis conducted with TAMELESS. The graph shown in Figure 7.14 illustrates the number of nodes added by the LLM for each analyzed case study.

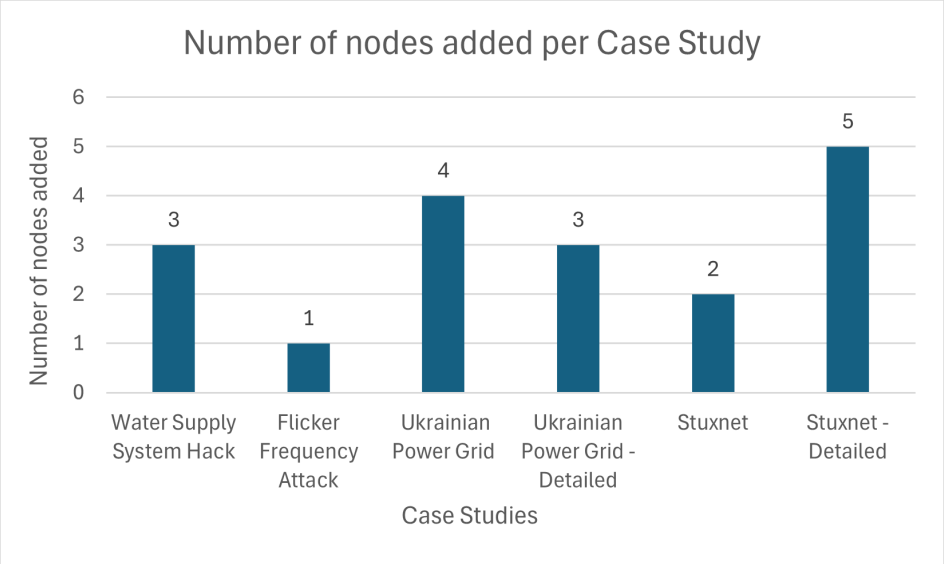


**Figure 7.14:** Number of nodes added per Case Study.

As highlighted by the graph, the integration of the LLM module provides a significant contribution to enriching the final analysis. In most cases, the model identifies relevant entities, actually involved in real-world cases, that were not considered in the formal analysis. For example, in the case of the "Water Supply System Hack", the model identifies the node related to the Remote Access Tool, while for the "Cyberattack on the Ukrainian Power Grid," nodes such as the Call Center System and the Uninterruptible Power Supply emerge, critical elements in

the attack's dynamics.

Other nodes, although less critical, may appear superfluous for the completion of the attack. However, they still contribute to a more comprehensive view of possible attack paths, also highlighting auxiliary techniques.

### Mitigation and Detection

Beyond adding new nodes, the new module enhances the analysis by leveraging the characteristics of MITRE ATT&CK. In addition to providing useful insights about the technique, it also associates a set of Mitigation techniques and Detection strategies. This information is fundamental for making the system resilient to identified threats.

For example, for the technique **Spearphishing Attachment (T1566.001)** in the case of the *Ukrainian Power Grid*, the module highlights that:

- **Mitigation:**
  - *User Training:* Train operators to recognize possible attacks or unauthorized access attempts to reduce the success rate of social engineering techniques.
  - *Restrict Web-Based Content:* Limit access to certain websites, block specific downloads, or prevent the opening of particular attachments.
  - *Antivirus/Antimalware:* Keep security solutions updated with signature-based and behavioral detection techniques to identify malicious attachments.
- **Detection:**
  - *Monitoring Email and Network Traffic:* Use intrusion detection systems to intercept anomalous patterns and typical signs of spearphishing based on signatures and behavior.
  - *Process Analysis:* Observe endpoint behavior to detect the execution of suspicious processes.

Overall, the obtained results highlight how the system, through this information, provides immediate operational support for incident identification and response.

This approach fully exploits the potential of MITRE ATT&CK, leveraging its contribution to attack prevention. It is important to note that these **Mitigation** and **Detection** techniques are not always available for all identified techniques. An example is *Loss of Protection (T0837)*, identified for the node Uninterruptible Power Supply. This occurs especially when the threat is too generic and represents a compromised system state, making it difficult to identify a specific attack that caused it.

# Chapter 8

# Conclusions and Future Developments

## 8.1 Conclusions

The primary goal of this thesis was to enhance and expand the hybrid threat analysis system named TAMELESS, by incorporating large language models with the MITRE ATT&CK framework, thereby enhancing the completeness and efficiency of threat analysis.

One of the main elements analyzed in the research was the formal verification of the derivation rules of the model using the PRISM tool, allowing for a formal check on the coherence of the system. The formal analysis confirmed that the rules defined in the model are consistent and do not contain any contradictions, and that the system is competent in correctly handling the transitions among the different security states.

Simultaneously, the integration of a Large Language Model (LLM) with Retrieval-Augmented Generation (RAG) technology made it possible to merge the methodological rigor characteristic of conventional analysis with a more responsive and adaptive one, without compromising the formal consistency of the model. This integration allowed the system to identify potential attack paths and even recommend techniques that could pose a threat to a given entity. It also relates these techniques to corresponding mitigation and detection strategies based on information from

the MITRE ATT&CK framework.

Another significant improvement involves the introduction of a user-friendly graphical interface, which makes interaction with the system easier for the user, allows for visualization of the attack graph as well as the different metrics introduced, and improves understanding of the analysis results. This enhancement has made TAMELESS more accessible, especially for users without a high level of expertise in the model.

In conclusion, this dissertation has demonstrated the validity of the suggested approach by verification on real-world case studies, including high-profile events such as "Stuxnet" and the "Cyber attack on the Ukrainian Power Grid". The results demonstrated that the new module complements the analysis process with more information regarding the nodes involved compared to the formal analysis, thereby enhancing the identification of attack pathways and making threat reduction recommendations.

## 8.2 Limitations and Critical Issues

The limitations identified during the thesis work will now be presented, highlighting some shortcomings that TAMELESS may have, even with the addition of the new model.

The first evident limitation is the Large Language Model used. LLaMA 3.1 at 8B is an efficient and lightweight model, excellent for various applications, but its performance cannot be compared to larger and more complex models. It was chosen because, despite the remarkable computing power of the IridisX platform [29], kindly provided by the University of Southampton, its computing nodes were isolated, preventing the direct download of a model onto the node. To address this issue, the model necessarily had to be small enough. As a result, the obtained responses were less precise, reasoning patterns were repetitive, and in some simulations, hallucinations also occurred.

A second aspect to explore further is the use of the MITRE ATT&CK dataset. Although it is a well-known and authoritative source for identifying vulnerabilities, it does not contain all the necessary information. Each company experiences different attacks, and representing all of them would be impossible, especially when

the threats targeting the system are new or highly complex. This study introduced the case study "Flicker Frequency Attack," demonstrating how a very specific attack is particularly difficult to represent accurately.

Another limitation is the formal verification of the rules correctness. Since these rules are created and formulated based on real examples, formally verifying the completeness of all rules in the system is more complex. An important step in this direction was taken using PRISM, which confirmed the consistency of the system rules.

Finally, the lack of a temporal aspect in the system has not been addressed or overcome. The analysis performed is static, meaning that the security state is evaluated at that exact moment without the possibility of modification. Thus, if a security architect, after analyzing TAMELESS results or the identified mitigation and detection techniques, decides to change the system structure, the entire analysis would need to be performed again from the beginning.

These limitations bring into focus the areas of the system that need additional work so that the analysis becomes more accurate and reliable.

## 8.3 Perspectives and Future Developments

Taking into account the system limitations identified, possible future developments are proposed that could further improve the analysis.

Since the current LLM model has limitations for such a complex task, it is suggested to modify it by incorporating a more powerful model. One option is to upgrade to the same LLaMA 3.1, but with the 70B token version, aiming to maintain a favorable trade-off between performance and execution time of the analysis.

Another proposed modification is the integration of additional databases containing threat information, to seamlessly complement MITRE ATT&CK and attempt to fill its gaps. A valid alternative could be the "Common Vulnerabilities and Exposures" (CVE) repository [30], which, in addition to improving model coverage, could improve risk assessment by leveraging the "Common Vulnerability Scoring System" (CVSS)[31].

Finally, the generated graph currently represents only the final security state of

the system entities without illustrating how these states were derived. Introducing a mechanism that shows the reasoning behind these security states, possibly by displaying the applied rules, would enable a more transparent and detailed analysis of the compromise path, providing valuable insights for evaluating the entire system.

The work performed demonstrates how combining formal methods and LLMs increases threat analysis without losing the strict and reliable rules, while also additionally offering a more comprehensive view of the threats targeting the system. The results have provided opportunities for new areas of research and development, with the goal of making TAMELESS an increasingly complete and accessible tool.

# Appendix A

# LLaMA Module Prompts

## A.1 MITRE ATT&CK Vulnerability Identification Prompt

```
prompt = f"""
    You are a cybersecurity expert using the MITRE
  ATT&CK framework to identify vulnerabilities.

   ### Node Information
   - Name: {node_name}
   - Type: {node_type}

   ### System Graph Context
   {graph_context}

   ### RAG Context:
   {context}
   ### Task

```

```
15  Identify potential vulnerabilities for this node based on its name, type, the known vulnerabilities indicated in the system graph context (e.g., dynamic properties such as 'vulnerable') and logical attack paths.
16
17  Your goals are:
18  1. Examine the System Graph Context for any dynamic properties that indicate the node is 'vulnerable', in that section you will find all the known vulnerabilities for the specific node, include the corresponding MITRE ATT&CK technique.
19      If the node is marked as vulnerable (for example, with an associated threat like DenialOfService for the node call_center_system), automatically include this vulnerability in your response with the corresponding MITRE ATT&CK technique.
20  2. Discover new vulnerabilities using Node Information, System Graph Context, and RAG Context analysis. Try to understand what the node name can represent in the System Graph Context.
21  Ensure that the proposed vulnerabilities are realistic and logically derived from the node's connections and type.
22
23  #### 1. Select MITRE ATT&CK Techniques Based on Node Type
24  Each node type has specific categories of vulnerabilities that apply to it. Focus only on relevant vulnerabilities:
25
26  - Physical nodes
```

```
27      - Susceptible to physical access, tampering, and
    sabotage.
28      - If part of an Industrial Control System (ICS),
    they can also be indirectly affected by cyber
    threats.
29      - Exclude purely cyber-based exploits (e.g.,
    software-based malware, phishing, etc.) unless a
    direct relationship justifies otherwise.
30
31    - Human nodes
32      - Targeted primarily through social engineering
    techniques (phishing, credential theft,
    impersonation, etc.).
33      - Can be exploited to gain access to cyber
    systems.
34      - Exclude vulnerabilities that apply only to
    software, networks, or hardware.
35
36    - Cyber nodes
37      - Vulnerable to software and network-based
    attacks (e.g., remote code execution, authentication
    bypass, malware, etc.).
38      - If connected to a physically accessible system,
    physical access may be an indirect factor.
39      - Exclude vulnerabilities that require direct
    human or physical manipulation.
40
41    - General Rule:
42      - Only assign vulnerabilities that logically
    apply to the node type.
43      - If an attack technique requires conditions that
    are not met, but is still applicable, include it
    without modifications.
```

```
44
45      #### 2. Use System Graph to Validate Attack Paths
46          - Identify realistic attack vectors using actual
    graph connections, analyze all the node's relations.
47          - If the node is connected to another node with
    known spread or vulnerabilities, prioritize and
    evaluate that threat.
48
49      #### 3. Ensure Relevant MITRE ATT&CK Techniques
50          - Select a diverse set of techniques across
    multiple tactics (e.g., Initial Access, Execution,
    Discovery).
51          - Avoid irrelevant techniques.
52          - If a technique has missing conditions but is
    still applicable, include it without modifications.
53
54      ---
55
56      Respond only with this strict JSON format and
    take inspiration from the example provided below:
57
58      ### Example:
59
60      {{
61        "response": [
62          {{
63            "Tactic": {{tactic}},
64            "Technique": {{technique}} - {{MITRE ID}},
65            "Description": {{Official MITRE
    Description}}
66          }},
67          ...
68        ]
```

```
69        }}
70
71        ### Your Response:
72        Always include the description and the MITRE ID,
    which refers to the unique identifier from the MITRE
    ATT&CK framework associated with the specific
    technique (e.g., T0837).
73        Do not include any other text. Respond with a
    valid JSON object.
74        The Description should not include your reasoning.
75        <<<END PROMPT>>>
76      """
```

## A.2 System Graph Vulnerability Validation Prompt

```
1  prompt = f"""
2      You are a cybersecurity expert. Analyze the
    following system structure and assess if the given
    vulnerability is applicable to the specified node.
3
4      ### Node Information
5      - Name: {node_name}
6      - Type: {node_type} (e.g., physical, cyber, human)
7
8      ### Vulnerability
9      - Tactic: {tactic}
10     - Technique: {technique}
11     - Description: {description}
12
13     ### System Graph Context
14     {graph_context}
```

```
15
16   Analyze the system structure and relationships.
     Your primary focus must be on determining if the
     vulnerability applies directly to the specified node
     based on its characteristics and dynamic properties.
     Consider other nodes only if there is a clear,
     direct relationship that creates an attack vector
     enabling the vulnerability on the target node.
17
18   ---
19
20   ### Understanding the System Graph Relationships
21   The relationships in the system graph determine how
     vulnerabilities propagate and affect nodes.
22   The system relationships are structured as follows:
23
24   - **Contain(A, B):** A contains B, representing
     system composition (e.g., a server contains data).
25   - **Control(A, B):** A controls B (e.g., a
     controller controls sensors).
26   - **Connect(A, B, C):** A connects B to C (e.g., a
     network connects two servers). *Note: Connect is
     unidirectional.*
27   - **Depend(A, B):** A depends on B for
     functionality (e.g., a database server depends on
     storage).
28   - **Check(A, B):** A verifies that B is functioning
     normally.
29   - **Replicate(A, B):** A is a replica of B,
     enabling redundancy.
30   - **Protect(A, B, T):** A protects B from threat T
     (e.g., a firewall protects a system from hacking).
```

```
 - **Monitor(A, B, T):** A monitors B for threat T (e.g., an IDS detects cyber-attacks).
 - **Spread(A, T):** A can propagate threat T (e.g., phishing can spread malware).
 - **PotentiallyVulnerable(A, T):** A could become vulnerable to threat T under certain conditions.

 ---

 ### Node Type Vulnerability Guidelines
 Each node type has specific categories of vulnerabilities that apply to it. Focus only on relevant vulnerabilities:

 - Physical nodes
   - Susceptible to **physical access, tampering, and sabotage**.
   - If part of an **Industrial Control System (ICS)**, they can also be indirectly affected by cyber threats.
   - **Exclude purely cyber-based exploits** (e.g., software-based malware, phishing, etc.) unless a direct relationship justifies otherwise.

 - Human nodes
   - Targeted primarily through **social engineering techniques** (phishing, credential theft, impersonation, etc.).
   - Can be exploited to gain access to cyber systems.
   - **Exclude vulnerabilities that apply only to software, networks, or hardware**.

```

```
  - Cyber nodes
    - Vulnerable to **software and network-based attacks** (e.g., remote code execution, authentication bypass, malware, etc.).
    - If connected to a **physically accessible system**, physical access may be an indirect factor.
    - **Exclude vulnerabilities that require direct human or physical manipulation**.

  ---

  ### Target Node Focus and Validation Rules for the Vulnerability
  1. Direct Applicability to the Target Node**
     - Assess whether the vulnerability logically affects a node of type {node_type} based solely on its own characteristics and dynamic properties.
     - If the vulnerability is clearly not applicable to the target node, mark `"Exploitable": "No"` and explain why (e.g., a vulnerability that requires social engineering is not applicable to a purely physical node).

  2. Evaluation of Direct Attack Vectors
     - Only consider other nodes if a direct, explicit relationship (from the system graph) enables the attack vector for the target node.
     - Do not infer indirect or speculative attack paths that involve nodes with no clear linkage to the target.

  3. Node Type Consistency
```

```
    - Validate that the vulnerability technique is inherently applicable to a node of type {node_type}.
    - For example, techniques requiring human interaction or social engineering should not be deemed applicable to nodes that are purely physical unless a direct relationship or additional property justifies it.

 4. Explanation Quality
    - Provide a concise, unique explanation that justifies why the vulnerability is or is not exploitable for the target node.
    - Include mention of any direct relationships that support exploitation if applicable.
    - Do not include your internal reasoning or extraneous text only the final determination and explanation.

 ---
 ### Task
 Determine if the vulnerability is applicable to the node and provide a reasoned response in the following strict JSON format:

 {{
   "response": {{
     "Exploitable": <Yes/No>,
     "Reason": <Brief explanation based on the node's characteristics, its dynamic properties, and any direct relationships that enable or prevent exploitation>
   }}
 }}
```

```
85
86      Do not include any other text. Respond with a valid
        JSON object.
87
88      <<<END PROMPT>>>
89      """
```

# Bibliography


[1] Fulvio Valenza, Erisa Karafili, Rodrigo Vieira Steiner, and Emil C. Lupu. «A Hybrid Threat Model for Smart Systems». In: *IEEE Transactions on Dependable and Secure Computing* 20.5 (2023), pp. 4403–4417. DOI: 10.1109/TDSC.2022.3213577 (cit. on pp. 2, 3, 9, 13, 24, 25, 27, 28).

[2] Nataliya Shevchenko, Timothy A. Chick, Paige O'Riordan, Thomas Patrick Scanlon, and Carol Woody. *Threat Modeling: A Summary of Available Methods*. Tech. rep. REV-03.18.2016.0; Distribution Statement A: Approved for Public Release; Distribution Is Unlimited. Software Engineering Institute, Carnegie Mellon University, July 2018 (cit. on pp. 5, 6).

[3] Fulvio Valenza. *TAMELESS: Threat & Attack Model Smart System*. https://github.com/FulvioValenza/TAMELESS. 2023 (cit. on pp. 9, 15).

[4] *XSB Prolog*. http://xsb.sourceforge.net/ (cit. on p. 9).

[5] *MITRE ATT&CK*. https://attack.mitre.org/ (cit. on p. 10).

[6] *MITRE ATT&CK Navigator github :* https://github.com/mitre-attack/attack-navigator/?tab=readme-ov-file (cit. on p. 11).

[7] *PRISM :* https://www.prismmodelchecker.org/ (cit. on pp. 11–13).

[8] *Neo4j*. https://neo4j.com/ (cit. on p. 14).

[9] José Guia, Valéria Gonçalves Soares, and Jorge Bernardino. «Graph Databases: Neo4j Analysis». In: *Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS 2017)*. Vol. 1. SCITEPRESS – Science and Technology Publications, Lda., 2017, pp. 351–356. ISBN: 978-989-758-247-9. DOI: 10.5220/0006356003510356 (cit. on p. 15).

[10] Kira Sam and Raja Vavekanand. *Llama 3.1: An In-Depth Analysis of the Next Generation Large Language Model*. Preprint. July 2024. DOI: 10.13140/RG.2.2.10628.74882. URL: https://www.researchgate.net/publication/382494872_Llama_31_An_In-Depth_Analysis_of_the_Next_Generation_Large_Language_Model (cit. on p. 16).

[11] Amazon Web Services. *Retrieval-Augmented Generation*. https://aws.amazon.com/what-is/retrieval-augmented-generation/ (cit. on p. 17).

[12] *Lanchain*. https://www.langchain.com/ (cit. on p. 17).

[13] *FAISS*. https://github.com/facebookresearch/faiss (cit. on p. 17).

[14] Pool Reinsurance. *Oldsmar Water Facility Attack: Post-Incident Analysis*. https://www.poolre.co.uk/terrorism-threat-publications/oldsmar-water-facility-attack-post-incident/. Accessed: 2025-03-17. 2021 (cit. on p. 19).

[15] Sean Tufts. *Attempted Florida Water Supply Tampering Underscores IoT/OT Security*. URL: https://www.optiv.com/insights/discover/blog/attempted-florida-water-supply-tampering-underscores-iotot-security (cit. on p. 20).

[16] Robert M. Lee, Michael J. Assante, and Tim Conway. *Analysis of the Cyber Attack on the Ukrainian Power Grid: Defense Use Case*. Tech. rep. Released March 18, 2016; TLP: White. E-ISAC and SANS ICS, 2016 (cit. on p. 21).

[17] Eyal Ronen and Adi Shamir. *Extended Functionality Attacks on IoT Devices: The Case of Smart Lights*. Invited Paper. 2016. URL: https://www.researchgate.net/publication/382494872_Extended_Functionality_Attacks_on_IoT_Devices_The_Case_of_Smart_Lights (cit. on pp. 22, 69).

[18] Marie Baezner and Patrice Robin. *Hotspot Analysis: Stuxnet*. Report. Oct. 2017. DOI: 10.3929/ethz-b-000200661. URL: https://doi.org/10.3929/ethz-b-000200661 (cit. on p. 23).

[19] *React Js*. https://react.dev/ (cit. on p. 38).

[20] *Tailwind CSS*. https://tailwindcss.com/ (cit. on p. 38).

[21] *Cytoscape.js*. https://js.cytoscape.org/ (cit. on p. 39).

[22] *Hugging Face, LLaMA 3.1 8b*. https://huggingface.co/meta-llama/Llama-3.1-8B (cit. on p. 50).

[23] *Sentence Transformer, all-MiniLM-L6-v2*. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2 (cit. on p. 51).

[24] Aobo Kong, Shiwan Zhao, Hao Chen, Qicheng Li, Yong Qin, Ruiqi Sun, Xin Zhou, Enzhi Wang, and Xiaohang Dong. «Better Zero-Shot Reasoning with Role-Play Prompting». In: *arXiv preprint* (2024). arXiv: 2308.07702 [cs.CL]. URL: https://arxiv.org/abs/2308.07702 (cit. on pp. 52, 53).

[25] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. «Chain-of-Thought Prompting Elicits Reasoning in Large Language Models». In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2022. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/8bb0d291acd4acf06ef112099c16f326-Paper-Conference.pdf (cit. on pp. 52, 53).

[26] Connor Shorten, Charles Pierse, Thomas Benjamin Smith, Erika Cardenas, Akanksha Sharma, John Trengrove, and Bob van Luijt. «STRUCTUREDRAG: JSON RESPONSE FORMATTING WITH LARGE LANGUAGE MODELS». In: *arXiv preprint* (2024). arXiv: 2408.11061 [cs.CL]. URL: https://arxiv.org/abs/2408.11061 (cit. on pp. 52, 54).

[27] Hamed Babaei Giglou, Jennifer D'Souza, and Sören Auer. «LLMs4OL: Large Language Models for Ontology Learning». In: *arXiv preprint arXiv:2307.16648* (2023). URL: https://arxiv.org/abs/2307.16648 (cit. on p. 54).

[28] National Institute of Standards and Technology. *Guide for Conducting Risk Assessments*. Tech. rep. SP 800-30 Revision 1. National Institute of Standards and Technology, 2012. URL: https://csrc.nist.gov/publications/detail/sp/800-30/rev-1/final (cit. on p. 61).

[29] *IRIDIS X - University of Southampton High-Performance Computing*. https://www.southampton.ac.uk/isolutions/staff/iridis (cit. on p. 95).

[30] MITRE Corporation. *Common Vulnerabilities and Exposures (CVE)*. 2024. URL: https://www.cve.org (cit. on p. 96).

[31] FIRST - Forum of Incident Response and Security Teams. *Common Vulnerability Scoring System (CVSS) v3.1 Specification Document*. 2019. URL: `https://www.first.org/cvss/v3-1/specification-document` (cit. on p. 96).