UNIVERSITÀ DEGLI STUDI ROMA TRE

Department of Civil, Computer Science and Aeronautical Technologies Engineering

Master's Degree Course in Computer Engineering

ADTViewer: a User Interface for the Analysis of Cyber Intrusions

By

Francesco Spezzano

ID 534050

Supervisor

Prof. Stefano Iannucci

Co-Supervisor

Dott. Tommaso Caiazzi

Academic Year 2023/2024

For those who matter most.

Acknowledgments

This thesis marks the conclusion of a journey that has lasted several years, filled with challenges, growth, and many moments of satisfaction. I would like to take this opportunity to express my gratitude to those who have supported me along the way.

First, I would like to thank my supervisor, Prof. Stefano Iannucci, for his guidance, support, and expertise, which have made this thesis not only possible but also an engaging and enjoyable experience.

I am also sincerely grateful to my co-supervisor, Dr. Tommaso Caiazzi, for his guidance throughout this project. In moments of uncertainty, he has always provided me with clarity and direction with patience and expertise.

In addition, I would like to thank my parents for their unwavering support and encouragement throughout this journey.

A special thank you goes to my brother, Matteo, your presence has meant more than words can express.

To Nina, my companion, thank you for your patience and encouragement throughout this journey. Your support has been my anchor, and I could not have done this without you.

Finally, to my friends, both those who have been with me from the beginning and those who have joined along the way, thank you for the countless moments of joy, laughter, and motivation. Your presence has made this experience truly special.

Contents

List of Figures

Abstract

This thesis presents ADTViewer, a plugin for OpenSearch Dashboards that provides an interactive graphical interface for intrusion analysis based on Attack Defense Trees. The system integrates with PANACEA, a framework that utilizes Timed Stochastic Games for the synthesis of optimal defense strategies. Through ADTViewer, users can load, visualize, and modify attack trees, explore optimal strategies, compare defense policies, and analyze the costs associated with different security scenarios. The system architecture was developed using a modular approach, combining technologies such as React, D3.js, Docker, and a backend infrastructure based on PostgreSQL and OpenSearch. The plugin is designed to ensure scalability and efficiency, reducing the complexity of attack analysis and improving intrusion response management. A case study demonstrates its effectiveness, comparing an unrestricted attack path with a mitigated scenario, showing how targeted defenses can impact cost and success rates. The results highlight how removing specific attack actions can significantly impact both costs and strategy effectiveness, demonstrating how ADTViewer advances theoretical security modeling into practical intrusion analysis while supporting strategic decision-making in cybersecurity.

Chapter 1

Introduction

1.1 Importance of Cybersecurity

In an era where digitalization is deeply embedded in all aspects of society, cybersecurity plays an important role in protecting sensitive information, ensuring system integrity, and maintaining operational resilience. The increasing reliance on interconnected systems, cloud services, and artificial intelligence has expanded both the attack surface and the complexity of potential threats. Organizations, governments, and individuals face an ever-evolving threat landscape, where cyber attacks can lead to financial losses, operational disruptions, and risks to data confidentiality.

To counter these threats, cybersecurity strategies must evolve beyond traditional defenses, integrating proactive risk assessment, real-time monitoring, and automated response mechanisms. Effective security frameworks should not only detect and prevent cyber attacks, but also provide structured methodologies to analyze vulnerabilities and anticipate future threats.

1.2 Challenges in Intrusion Response and Attack Tree Analysis

Intrusion response is a fundamental aspect of cybersecurity, aimed at detecting, mitigating, and preventing security incidents in real time. As cyber threats become more sophisticated, organizations must adopt structured methodologies to assess risks, anticipate attacker behavior, and deploy effective countermeasures.

An effective way to approach intrusion response can be achieved through Attack Trees (ATs) and their extension, Attack-Defense Trees (ADTs). Attack trees provide a hierarchical representation of how an attacker might attempt to compromise a system, while attack-defense trees extend this model by incorporating defensive actions that can mitigate or neutralize threats. By systematically analyzing attack paths and potential countermeasures, ADTs serve as a foundation for intrusion response planning, enabling security teams to model, evaluate, and optimize response strategies against evolving cyber threats.

Despite their effectiveness, attack trees present significant challenges when applied in practical cybersecurity contexts. The complexity of modern digital infrastructures leads to attack trees of increasing size and complexity, making manual analysis inefficient and error-prone. Traditional methods often rely on static conditions, not taking into account the dynamic nature of cyber attacks, where attackers adapt their strategies in response to implemented defenses.

1.2.1 Limitations of Existing Solutions

Several existing solutions for attack tree analysis present significant limitations that prevent their adoption in cybersecurity operations. ADTool, for instance, offers a static visualization of attack trees, which does not allow users to dynamically explore attack paths or interactively analyze mitigation strategies. PRISM-Games, a probabilistic model checker for security policies, provides powerful verification tools but requires users to manually define attack-defense models using complex formal descriptions, making it

a challenge to use effectively.

Another major limitation is the lack of integration with security monitoring platforms as many of these tools function as standalone applications, without direct integration into Security Information and Event Management (SIEM) systems, thereby reducing their ability to correlate modeled attack scenarios with real-time threat intelligence data.

These limitations highlight the need for an accessible and interactive solution that integrates attack tree analysis with modern cybersecurity frameworks, enabling security professionals to visualize, interact with, and modify attack scenarios in real time.

1.2.2 The need for a GUI for attack tree analysis

To overcome the challenges associated with current solutions, there is a strong need for a graphical, interactive tool that makes attack tree analysis more accessible and practical for real-world cybersecurity applications. A dedicated Graphical User Interface (GUI) would provide several key advantages in this context.

A well-designed GUI would significantly improve usability, allowing security analysts to interact visually with attack trees, explore different attack-defense scenarios dynamically, and analyze security risks. Unlike static tools, an interactive interface could incorporate real-time policy computation, enabling users to automatically generate and compare optimal attack-defense strategies based on different attack scenarios.

The integration of a GUI with SIEM and security monitoring systems, such as OpenSearch Dashboards, would enhance real-time cybersecurity operations by allowing security professionals to correlate detected threats with modeled attack trees, bridging the gap between theoretical risk assessment and active intrusion response. In addition, by including cost-benefit analysis tools, such a system could allow organizations to evaluate the trade-offs between different mitigation strategies.

1.2.3 Motivation and Objectives of the Thesis

The motivation behind this research is to address the limitations of current attack tree analysis tools by developing an interactive solution that enhances usability and integration with modern security frameworks. Many existing platforms require users to operate through command-line interfaces or complex scripting environments, making them inaccessible to professionals without advanced technical expertise. Additionally, these tools often lack real-time visualization support, making it difficult to interpret and analyze security scenarios effectively.

This thesis introduces ADTViewer, an advanced GUI that enhances the usability of attack tree analysis. The system is built on a modular architecture, integrating technologies such as React, D3.js, OpenSearch, and Docker to ensure seamless and responsive interaction. In addition, by incorporating the PANACEA framework, which utilizes Timed Stochastic Games to generate optimal defense strategies, the tool allows for automated synthesis of security policies based on dynamic attack scenarios.

1.3 Structure of the Thesis

The thesis begins by establishing the fundamental concepts in Chapter 2, where the principles of attack trees and attack defense trees are introduced as essential tools for cybersecurity risk assessment. The chapter explores their role in intrusion response systems, discusses existing attack tree analysis tools such as ADTool and PRISM-Games, and introduces PANACEA, a framework that integrates timed stochastic games to optimize security policies. These theoretical foundations are crucial for understanding the methodologies developed in later chapters.

In Chapter 3, the focus shifts to the system architecture of the ADTViewer plugin, detailing its integration with OpenSearch Dashboards. The chapter examines the modular front-end structure, built with React, and the backend implementation, which facilitates seamless communication between PANACEA, OpenSearch, and PostgreSQL databases. Additionally, the chapter discusses the containerized deployment

using Docker, ensuring scalability and ease of integration into existing security infrastructures.

Building on this, Chapter 4 provides a comprehensive user guide, illustrating the workflow of the ADTViewer plugin. It explains how users can load attack trees, compute and visualize optimal attack-defense strategies, and modify security scenarios by excluding specific actions. The chapter also presents the cost analysis and policy comparison functionalities, which enable security analysts to evaluate different mitigation strategies and make informed decisions based on monetary and operational trade-offs.

In Chapter 5, the thesis demonstrates the effectiveness of the developed system through a case study involving two distinct attack-defense scenarios. The first scenario examines an unrestricted attack path, where the attacker follows an optimal strategy without interference. The second scenario introduces a mitigation strategy, removing a key attack step and analyzing its impact on the overall attack success rate and associated costs. This chapter highlights how the ADTViewer plugin can be used to quantify the impact of security interventions and improve decision-making in real-world cybersecurity operations.

The thesis concludes by summarizing the contributions made, both theoretical and practical. It discusses the limitations and challenges encountered during the research and suggests future improvements, such as enhancing computational efficiency for large attack trees, integrating machine learning techniques for predictive security analysis, and developing adaptive security policies capable of dynamically responding to emerging cyber threats.

Chapter 2

Background and Related Work

Cybersecurity is a field that continuously evolves to address new threats and vulnerabilities. One of the methodologies used to analyze security risks is the concept of Attack Trees (ATs), which provide a hierarchical representation of attack strategies and facilitate a systematic security assessment. This chapter provides an overview of attack trees and their extensions, particularly Attack Defense Trees (ADTs). Additionally, we review existing tools for attack tree analysis and explore the PANACEA framework, which leverages PRISM-Games for optimizing security strategies.

2.1 Attack Trees

Attack Trees are a structured way to model the security of systems by hierarchically representing different attack strategies. The name was first mentioned by Salter et al. in 1998 [SSSW98] but is often only attributed to Schneier [Sch99]. By breaking down complex attack strategies into smaller components, ATs help identify vulnerabilities, assess risks, plan mitigation, and improve security.

2.1.1 Structure of Attack Tree

In the attack tree formalism, the main goal of an attacker is depicted as the root node of a tree, the goal is then disjunctively or conjunctively refined into sub-goals. The

refinement is repeated recursively, creating a tree-like structure until the reached node represents basic actions.

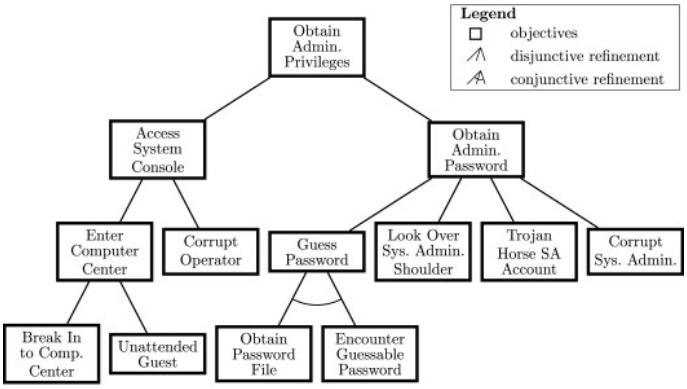

Figure 2.1: Example of an Attack Tree from [Wei91]. Obtaining administrator privileges on a UNIX system.

The components of an attack tree, as seen in Figure 2.1 are:

- **Root Node**: Represents the main goal of the attacker.
- **Intermediate Nodes**: Denote the different strategies or paths that an attacker may take. They may require the accomplishment of several preceding steps.
- **Leaf Nodes**: Represent the specific steps or conditions necessary for an attacker to reach their objective or an intermediate goal.
- **Edges**: The connections between nodes, indicating the flow and dependency of actions.
- **Conjunctive refinement** *(AND Gate)*: All sub-goals linked by an AND gate must be completed to achieve the parent goal.
- **Disjunctive refinement** *(OR Gate)*: Any one of the sub-goals linked by an OR gate is sufficient to fulfill the parent goal.

2.1.2 Applications in Cybersecurity

Attack trees can be used to analyze security threats by systematically breaking down attack strategies. Their structured approach has led to various cybersecurity applications, such as risk assessment for critical infrastructure security and security analysis of IoT devices. In the case of IoT security, attack trees are used to evaluate vulnerabilities in connected devices [XAH+16]. Another relevant example is the assessment of vulnerabilities in healthcare devices, where attack trees have been used to evaluate different threat scenarios and design appropriate countermeasures[SSH+18].

In conclusion, the structured methodology of attack trees enables organizations to visualize attack pathways, assess risks effectively, and implement well-informed countermeasures, ultimately strengthening system security [NFW17].

2.1.3 Attack Defense Trees

Attack Defense Trees (ADTs) are an extension of traditional attack trees that incorporate defensive actions to model the interplay between attackers and defenders [KMRS14]. While attack trees focus exclusively on the strategies available to an attacker, ADTs introduce defense nodes that represent security measures designed to mitigate or prevent attacks. This approach enables the identification of optimal defense strategies and minimization of security risks across various domains, such as cyber-physical systems and cloud security. As illustrated in Figure 2.2, ADTs provide a structured representation of attack scenarios by integrating both offensive and defensive actions, allowing for a comprehensive security analysis.

ADTs have been widely applied in cloud security, where they help model potential cyber threats and determine effective countermeasures. For example, [KMRS12] demonstrated how ADTs can be used to analyze multi-layered cloud security models, allowing for a systematic evaluation of vulnerabilities and defensive solutions. Similarly, in critical infrastructure security, ADTs have been used to model cyber threats against essential systems like power grids and financial institutions [BFP06].

ADTs have also been integrated with timed automata and Bayesian networks to

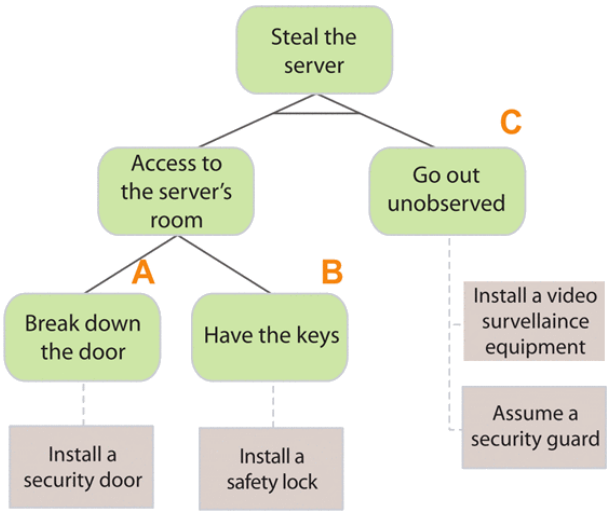

Figure 2.2: Example of an Attack Defense Tree representing a server theft [BFP06].

enhance predictive capabilities. Timed automata help model time-dependent attack and defense strategies, crucial in dynamic environments [GHL+16], meanwhile, Bayesian networks enable probabilistic risk assessment, allowing to evaluate the likelihood and impact of different attack scenarios [RRI+20].

2.2 Existing Tools for Attack Tree analysis

The increasing importance of attack trees and attack defense trees in cybersecurity has led to the development of various tools that aid in their modeling, analysis, and risk assessment. These tools provide functionalities such as graphical modeling, probabilistic risk assessment, and game-theoretic analysis.

2.2.1 ADTool

ADTool[1] is an open-source software developed at the University of Luxembourg for graphical modeling and quantitative analysis of security scenarios using ADTs. It provides an intuitive GUI for model creation, modification, and evaluation, ensuring com-

[1] `https://satoss.uni.lu/members/piotr/adtool/`

pliance with the ADTree graphical language. The tool employs an improved version of Walker's algorithm [II89] for optimized tree layouts and displays the corresponding attack-defense term in real time. It supports advanced model manipulation features such as folding, expanding, zooming, and hiding tree sections, making it useful for analyzing large models and facilitating presentations. ADTool includes a bottom-up evaluation algorithm for various attributes (e.g., cost, probability, time), supporting attacker, defender, and combined perspectives. An overview table ensures consistency, particularly in large models. Having been extensively tested, ADTool efficiently handles thousands of nodes, with computations performed instantly [KKMS13].

2.2.2 PRISM and PRISM-Games

PRISM[2] is a well-known probabilistic model checker widely used for analyzing stochastic models, including Markov Decision Processes [KNP11]. PRISM-Games extends PRISM to handle multi-player stochastic games, making it particularly useful for cybersecurity applications where attackers and defenders interact strategically [FKN+20]. Key features of PRISM and PRISM-Games:

- **Formal modeling**: Enables the representation of attack-defense interactions as stochastic games, allowing security analysts to model adversarial scenarios effectively.
- **Automated verification**: Performs quantitative security analysis, calculating attack success probabilities and associated costs.
- **Optimal strategy synthesis**: Supports the computation of optimal attack and defense strategies, helping organizations design cost-effective security measures.
- **Multi-objective evaluation**: Allows users to analyze trade-offs between attack probability, cost, and system resilience.

[2] `https://www.prismmodelchecker.org/`

PRISM and PRISM-Games have been successfully applied in critical infrastructure security, IoT protection, and cloud security [BCG18], providing a formal verification framework for security risk assessment.

2.3 PANACEA: Attack Tree analysis with PRISM-Games

PANACEA [Mar23] is a framework designed to enhance Intrusion Response Systems by automating the generation of optimal defense policies against cyber threats. PANACEA addresses the limitations of traditional IRS approaches by combining ADTs with Timed Stochastic Games, allowing defenders to model cyberattacks as structured decision-making problems.

Using PRISM-Games, the framework transforms an attack-defense model into a game-theoretic representation, where both attackers and defenders make strategic choices. By applying this approach, PANACEA systematically evaluates attack success probabilities and identifies cost-effective countermeasures, improving system security while optimizing resource allocation.

2.3.1 XML Attack Tree processing

A key step in PANACEA's workflow is processing ADTs, which serve as the input representation of the attack scenarios. The attack tree data, stored in XML format (compatible with ADTool), is processed to extract relevant attack paths, assign cost and execution time attributes to each attack step, and convert the structured attack-defense scenario into a Timed Stochastic Game.

This structured transformation allows PANACEA to incorporate time constraints and probabilistic success rates, enabling a dynamic evaluation of threats, where attack and defense actions have associated probabilities, costs, and response times, unlike traditional static analyses.

2.3.2 PRISM-Games and Optimal Policies

Once the attack-defense model is transformed into a formal game representation, PANACEA uses PRISM-Games for probabilistic verification and strategy synthesis. By analyzing attack scenarios, PRISM-Games computes the probability of an attack succeeding under different security policies and identifies the optimal defensive actions to minimize risk while considering cost and feasibility.

The result, illustrated in Figure 2.3, is a defense policy that determines the best sequence of actions based on system state and observed attacker behavior. These policies account for timing constraints and resource availability, ensuring that countermeasures are both effective and operationally feasible.

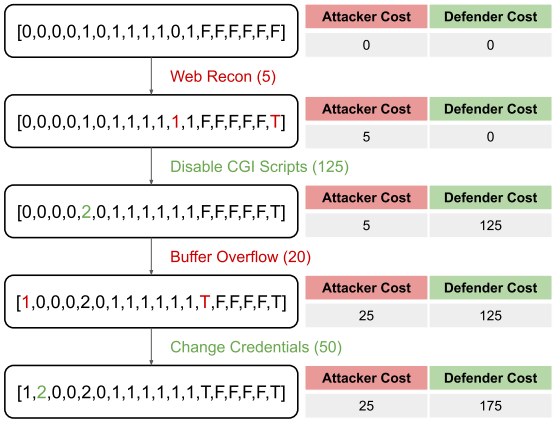

Figure 2.3: Example of an optimal strategy computed by PANACEA not including time.

2.4 Intrusion Response Systems

Intrusion Response Systems (IRS) are important security components designed to detect, analyze, and mitigate intrusions in real time. These systems monitor network traffic, system logs, and behavioral patterns to identify suspicious activities and initiate appropriate countermeasures, such as blocking connections or alerting administrators. Early IRS relied mainly on rule-based detection methods, which proved inadequate to

address novel or evolving threats [SSEJJD12]. In contrast, modern IRS leverage advanced technologies such as machine learning and artificial intelligence to detect anomalies and respond to sophisticated cyberattacks in real time [AIA21].

A key challenge in IRS development is to balance detection accuracy with response speed. High false positive rates can flood system administrators with alerts, leading to response fatigue, while excessive false negatives can allow critical security breaches to go undetected [AMZZ+17]. To address these issues, current research focuses on integrating adaptive learning mechanisms and automated decision making processes into IRS frameworks, enabling them to dynamically adjust to emerging threats and improve overall resilience [GKM23].

2.5 Security Information and Event Management (SIEM) Solutions

In modern cybersecurity architecture, Security Information and Event Management (SIEM) systems play an important role in log management, threat detection, and incident response. These solutions collect security-related data from various sources and provide organizations with centralized visibility and real-time analysis to efficiently detect and respond to security threats [PR19]. There are several SIEM solutions available on the market, ranging from commercial platforms such as Splunk or Microsoft Sentinel to open source alternatives such as Wazuh and the Elastic Stack [TMA+22].

Among the various SIEM options, Wazuh[3] has emerged as an open source alternative that offers comprehensive capabilities for threat detection, log analysis, and file integrity monitoring. A key advantage of Wazuh is its ability to integrate seamlessly with OpenSearch, allowing the storage and visualization of security data effectively. As illustrated in Figure 2.4, this integration enhances the ability to analyze security logs, detect anomalies, and respond to potential threats in a structured and scalable way [TRP+24].

[3] `https://documentation.wazuh.com/current/index.html`

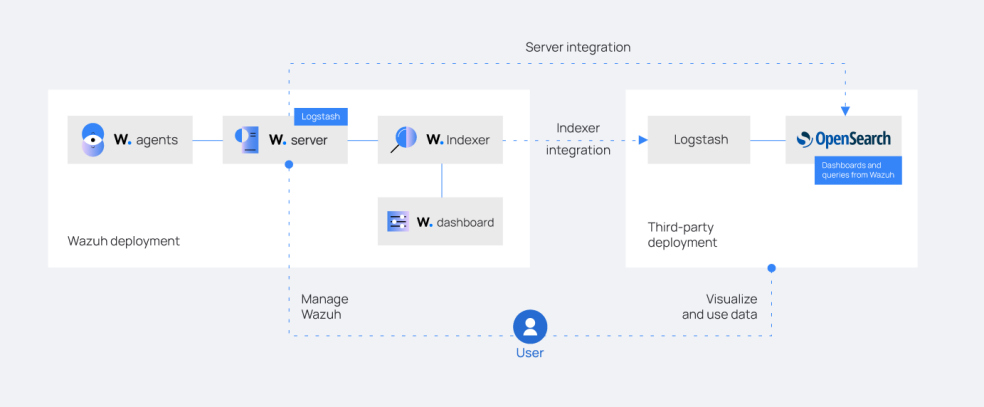

Figure 2.4: OpenSearch integration with Wazuh from the Wazuh documentation [3].

2.6 Towards a Graphical Interface for Attack Tree Analysis

The methodologies discussed in this chapter highlight the importance of structured threat modeling in cybersecurity. Attack trees and attack defense trees provide a systematic way to evaluate security risks and plan defensive measures. The reviewed tools, including ADTool and PRISM-Games, demonstrate the potential of automated analysis in optimizing security strategies. Moreover, the PANACEA framework shows how game-theoretic approaches can enhance Intrusion Response Systems by integrating probabilistic verification and strategic decision making.

However, accessibility remains a challenge, as many tools require command-line usage or scripting. To address this, the development of a user-friendly graphical interface is a necessity. The next steps in this research involve the design and implementation of such a graphical interface with the aim of bridging the gap between theoretical security analysis and practical applications.

Chapter 3

System Architecture

In this chapter, we present the integration of OpenSearch and OpenSearch Dashboards within the project, focusing on their roles in data visualization and security monitoring. We explore the development of a custom OpenSearch Dashboards plugin designed to provide an interactive GUI for the PANACEA framework, detailing its architecture and implementation. Additionally, we describe the containerized system architecture utilizing Docker and Docker Compose, explaining how various components interact within a multi-container environment. Finally, we examine the modular front-end architecture of the plugin, highlighting key components and their functionalities in managing and visualizing security policies and attack defense trees.

3.1 OpenSearch and OpenSearch Dashboards

OpenSerach[1] is a distributed search and analytics engine derived from the open source Elasticsearch[2] project and is widely used for log analytics, security monitoring and data visualization.

One of OpenSearch's main componets is OpenSearch Dashboards, a visualization tool that allows users to explore and analyze data stored in OpenSearch indices, providing a web-based interface for creating dashboards, running queries, and performing

[1] `https://opensearch.org/docs/latest/about/`
[2] `https://www.elastic.co/docs`

security analytics. It also includes built-in security features, such as role-based access control (RBAC), anomaly detection, and alerting. As stated in section 2.5 OpenSearch is fully integrable with Wazuh, allowing admins to identify trends, detect anomalies, and mitigate threats in real time.

3.1.1 Developing Plugins for OpenSearch Dashboards

OpenSearch Dashboards supports a flexible plugin architecture that allows developers to extend its functionalities by introducing custom visualization components, integrating with third-party security tools, or automating security workflows. A high-level overview of the OpenSearch Dashboards plugin architecture is shown in Figure 3.1.

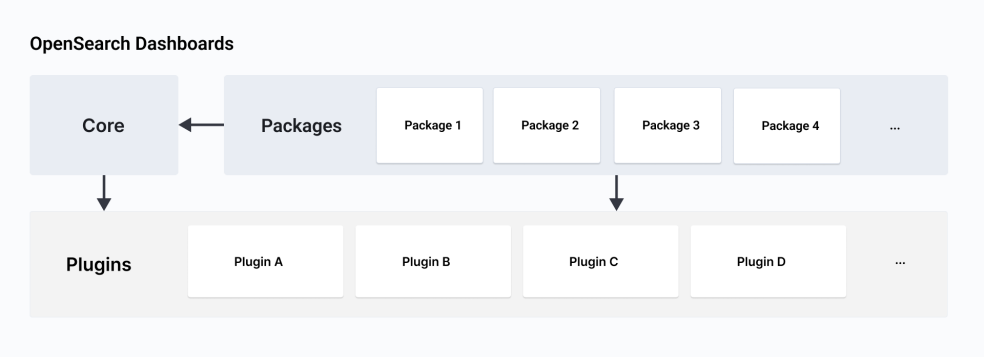

Figure 3.1: OpenSearch Dashboards plugin architecture [Pro22].

During this project, a custom plugin was developed to provide a graphical interface (GUI) for the PANACEA framework, enabling interactive visualization and management of attack defense trees. The plugin communicates with the PANACEA back end, retrieving security policies and attack tree data to display them in a useful format. The development of an OpenSearch Dashboards plugin involves the following steps:

- **Setting up the development environment**, including the installation of necessary dependencies and the OpenSearch Dashboard plugin framework.
- **Defining the plugin architecture**, which follows a modular design consisting of front-end components built with React and back-end services that manage API

integrations.

- **Integrating with the PANACEA back end**, exposing its core functionalities through custom APIs to facilitate interaction with the plugin.

3.2 Containerized System Architecture using Docker

One of the primary reasons for adopting Docker[3] in this project is that OpenSearch Dashboards provides an official[4] Docker image specifically for plugin development, simplifying the setup for testing and integrating custom plugins. Additional advantages of using Docker include its efficient resource utilization and rapid deployment capabilities. Additionally, Docker Compose offers a structured way to orchestrate multiple containers, facilitating communication between them, and allowing efficient network configuration.

3.2.1 Overview of the Multi-Container Setup

The system is structured as a multi-container architecture using Docker Compose for orchestration. The official OpenSearch Dashboards Docker Compose configuration was used as a foundation, extended to integrate additional containers required for system functionality. All containers are deployed within a shared network, ensuring efficient communication between services. As illustrated in Figure 3.2 the architecture consists of the following key components:

- **OpenSearch Node**: This container runs OpenSearch, serving as the primary search and indexing engine. It exposes a REST API that enables interaction with indexed data and facilitates query execution.

- **OpenSearch Dashborad Development Environment**: This container provides a pre-configured development environment for OpenSearch Dashboards, al-

[3] `https://www.docker.com/`
[4] `https://github.com/opensearch-project/OpenSearch-Dashboards`

lowing developers to build, test, and debug custom plugins without requiring a full OpenSearch installation.

- **PostgreSQL Database**: This container acts as the structured data repository for the system, storing information such as attack tree definitions and computed policies.

- **PANACEA Application**: This container hosts a streamlined version of PANACEA along with its dependencies. It is responsible for processing security policies, analyzing attack defense trees, and interacting with OpenSearch Dashboards and the PostgreSQL database.

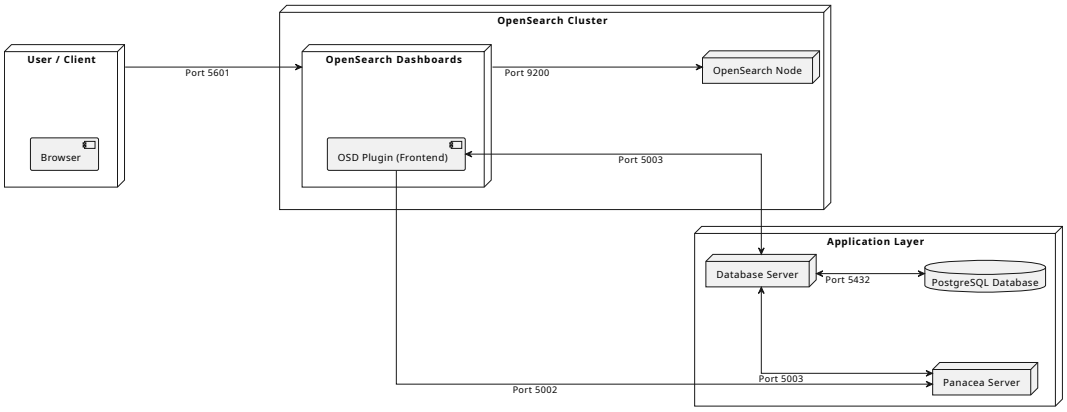

Figure 3.2: Deployment diagram for the multi-container setup.

3.3 Plugin Architecture

An OpenSearch Dashboard plugin follows the typical architecture of modern web applications, where the front end handles the user interface (UI) and interactions, and the back end manages server-side logic and the plugin's interaction with OpenSearch. The primary responsibilities of the front end, built using TypeScript and React, include the following:

- Creating pages and UI components.
- Defining routes to enable navigation within the plugin.
- API calls to the back end to retrieve and send data.

The back end of the plugin runs on the OpenSearch server side, allowing it to:

- Register REST API routes to receive requests from the front end.
- Access OpenSearch log data.
- Manage authentication logic and access plugin configuration settings.

3.4 Modular Front-end architecture

The ADTViewer[5] plugin for the PANACEA GUI is designed with a modular front-end architecture, ensuring ease of maintenance and scalability. The plugin consists of a main component which serves as the entrypoint, and several UI components that handle different functionalities. In the main component, data is fetched by calling the APIs defined in the back end, and the retrieved data is passed as props to other components as needed.

3.4.1 Main Component

The main component is responsible for organizing the various panels that make up the plugin's functionalities. Upon loading, it retrieves data from the database by making REST API calls defined in the back end that interact with the database server which exposes CRUD operations. Once the data is loaded, the other components (panels) are rendered and data is passed as props. To allow seamless interaction between the components and facilitate data sharing, a component serving as a context provider wraps all the others, ensuring that updates propagate efficiently across the interface.

[5]Repository available at: `https://github.com/Francsco99/adt_viewer`

The layout of the main component is fully configurable and structured to optimize workflow efficiency. The class diagram 3.3 illustrates the internal structure of the main component, highlighting the relationships between the core classes. The class `AdtViewerApp` serves as the central hub, initializing all other components. It maintains state variables such as `treeData`, `states`, `selectedPolicy`, and `selectedTree`, which are used to manage the data flow. The `TreeContextProvider` facilitates shared state management, ensuring consistent updates across the interface. The `Toolbar` component provides functionalities for uploading files and managing policies. The `TreeStateNavigator`, `NodeInfo`, and `StatesVisualizer` components are responsible for rendering and interacting with tree data. The `ActionsManager` handles user interactions related to policy management, while the `CostChart` and `PolicyComparisonChart` components generate visual analytics for policy evaluation.

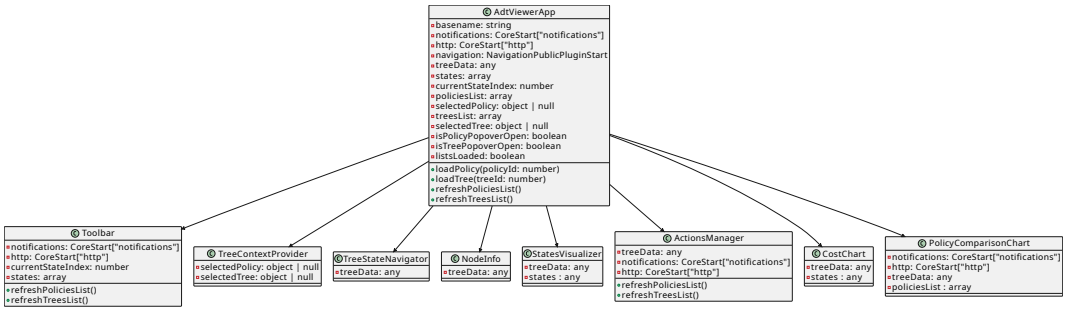

Figure 3.3: Class diagram for main component.

The sequence diagram 3.4 shows the interaction between the front end and back end. It begins when the user selects a policy or a tree, triggering an API request defined on the server side of the plugin. The back end processes this request by retrieving the corresponding data from the PANACEA server's database and returning the policy or tree to the front end. This interaction is supported through a Flask-based API running on the PANACEA server, which is responsible for handling requests and managing the database. The Flask application uses SQLAlchemy to interface with a PostgreSQL database and exposes multiple API endpoints to access trees, policies, and related data structures. Once the backend receives the requested information, it forwards the re-

sponse to the front end, where the `TreeContextProvider` updates the global state. This ensures that all relevant components, such as `TreeStateNavigator`, `NodeInfo`, and `StatesVisualizer`, are synchronized and reflect the newly loaded data in the user interface.

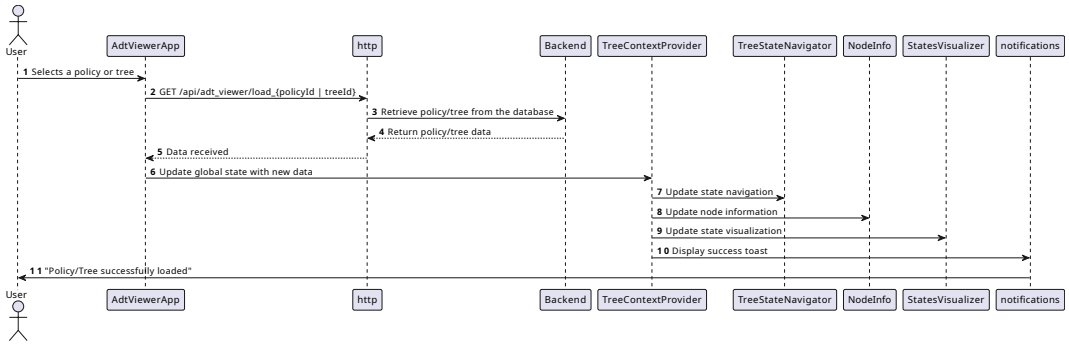

Figure 3.4: Sequence diagram of main component operations.

3.4.2 TreeContextProvider Component

The `TreeContextProvider` component's primary function is to manage and share a global state across the other UI components, ensuring seamless communication within the application. Unlike other components, `TreeContextProvider` does not render any UI elements; instead it acts as a centralized store for essential information such as selected and active nodes, policies, and trees. It also allows the other UI components to access and edit shared data without requiring a direct relationship, improving modularity and maintainability.

This architecture is crucial to enable dynamic updates across the UI, ensuring that any modification (such as selecting a node or changing a state) made in one panel is immediately reflected across the entire interface.

3.4.3 TreeVisualizer and TreeStateNavigator Components

The `TreeVisualizer` and `TreeStateNavigator` components work together to manage and display the attack defense tree, also allowing the user to cycle through its different

states.

The `TreeStateNavigator` functions as a control interface, allowing users to navigate different policy states while maintaining synchronization with the global state. Instead of directly managing the visualization, it delegates that responsibility to the `TreeVisualizer`, thus ensuring a clear separation between control logic and graphical representation. On the other hand, the `TreeVisualizer` is responsible for rendering the hierarchical attack tree structure, using D3.js[6] to handle layout calculations, node positioning, and interactive elements. All necessary data is retrieved from the `TreeContextProvider`, ensuring that changes (such as selecting a node or switching states) are immediately reflected across the UI.

3.4.3.1 Data Flow and State Management

As mentioned in Section 3.4.2, the `TreeContextProvider` ensures that `TreeStateNavigator` and `TreeVisualizer` remain loosely coupled while maintaining a consistent representation of the system state. The sequence diagram in Figure 3.5 details the interaction between the user and the component.

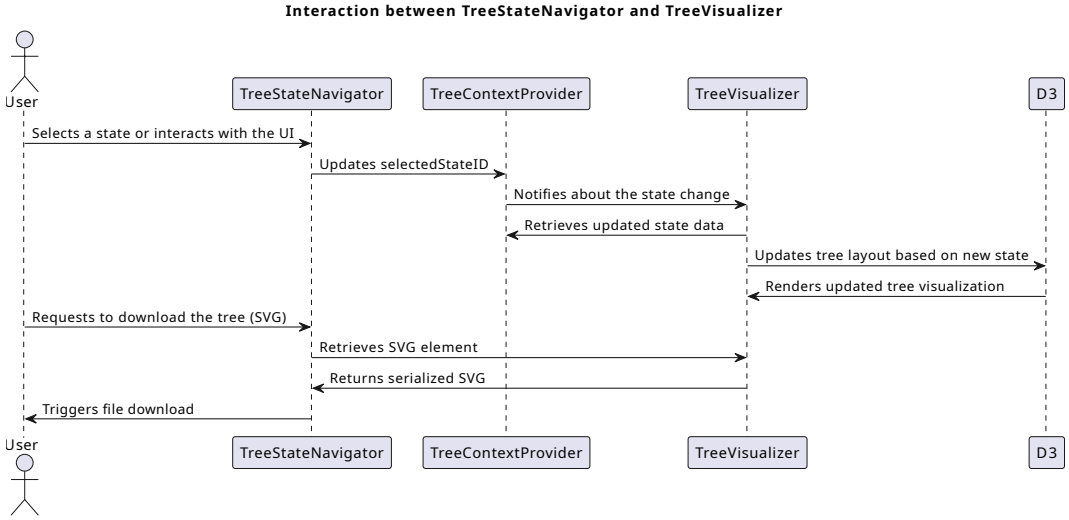

Figure 3.5: Sequence diagram for state selection and visualization updates.

[6] `https://d3js.org/`

3.4.4 StatesVisualizer Component

The `StatesVisualizer` component is responsible for displaying and managing the different states of the policy currently selected. This panel provides users with an overview of the tree's states, allowing them to analyze how each state evolves based on the applied policy. This visualization helps users better understand how the policy affects the system and allows them to determine the associated cost at each state transition. The class diagram in Figure 3.6 illustrates the internal structure of the component and its dependencies.

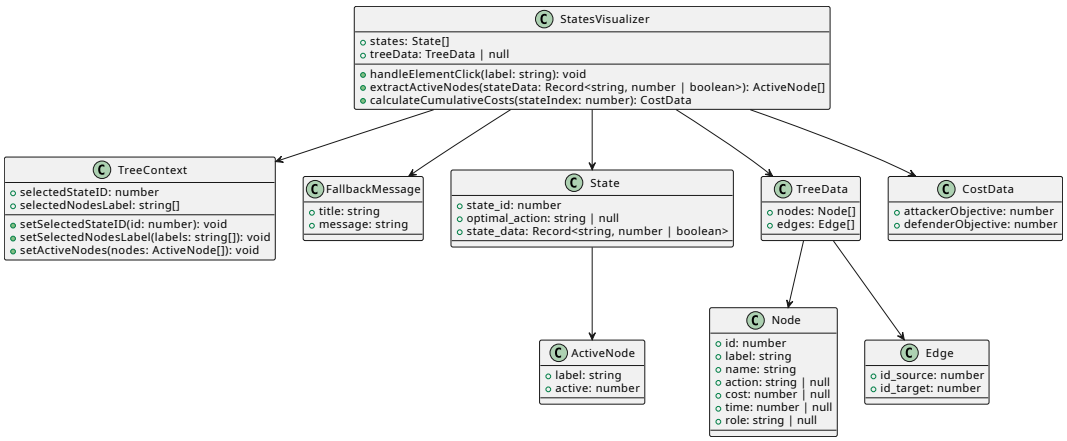

Figure 3.6: Class diagram for `StatesVisualizer` component.

3.4.5 ActionsManager Component

The `ActionsManager` panel allows user to interact with PANACEA by requesting a new policy computation for the tree currently selected. This panel displays all available actions and allows users to flag certain ones, excluding them from the policy computation. In that way, any flagged actions, along with its entire subtree, are omitted from further evaluation. The panel also provides sorting and filtering options, allowing users to refine the displayed actions based on their attributes, such as role, monetary cost, and time cost.

Once the desired actions have been flagged, the user can trigger the export process, which generates an updated representation of the tree structure and communicates with PANACEA to perform policy recalculations.

3.4.5.1 Export Configuration Process

When a user clicks the `Export Configuration` button, the system generates a JSON file representing the tree structure. Within this representation, flagged actions are marked as `hidden`, ensuring that they are ignored in the following policy computation. This JSON file is then sent to the PANACEA server, where it is processed and converted into the corresponding XML format required for PRISM-Games. After the new policy is calculated, PANACEA returns the updated results to the front end, where they are displayed to the user. A sequence diagram for this interaction is shown in Figure 3.7

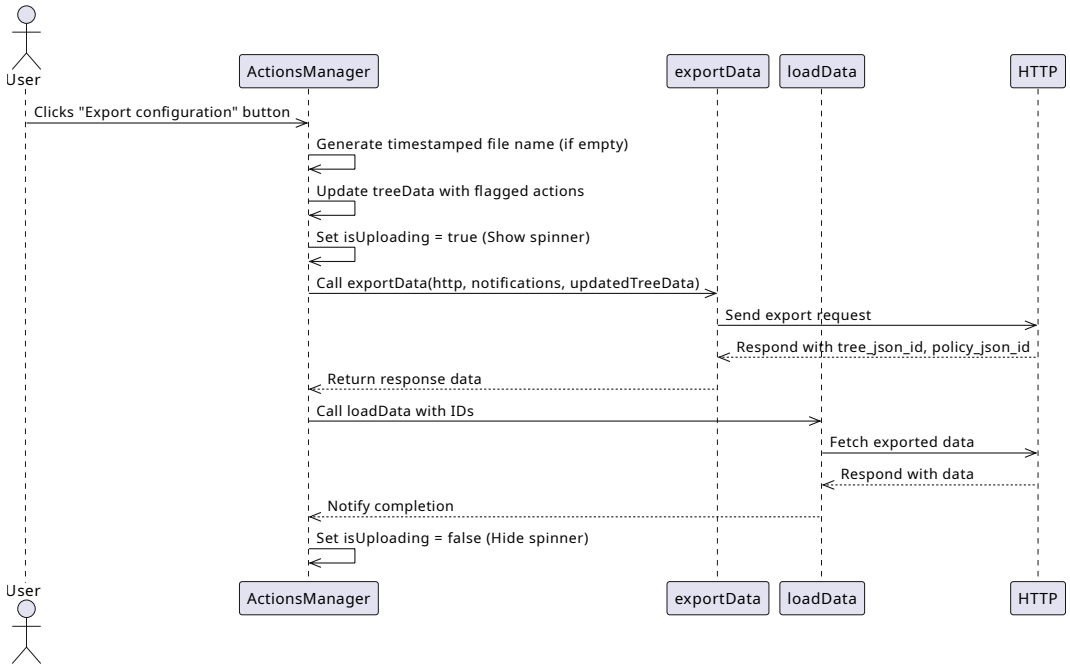

Figure 3.7: Sequence diagram showing the `Export Configuration` button interaction.

3.4.6 CostChart and PolicyComparisonChart Components

The `CostChart` and `PolicyComparisonChart` components are responsible for visualizing cumulative costs and comparing different policies within the system.

The `CostChart` component illustrates the evolution of cumulative costs over time, distinguishing between costs attributed to the attacker and the defender. The visualization is based on the sum of monetary and time costs associated with the optimal actions selected in each state. Data updates occur whenever a new state is selected, dynamically updating the graph to reflect values up to the current simulation point. This process is illustrated in the sequence diagram 3.8:

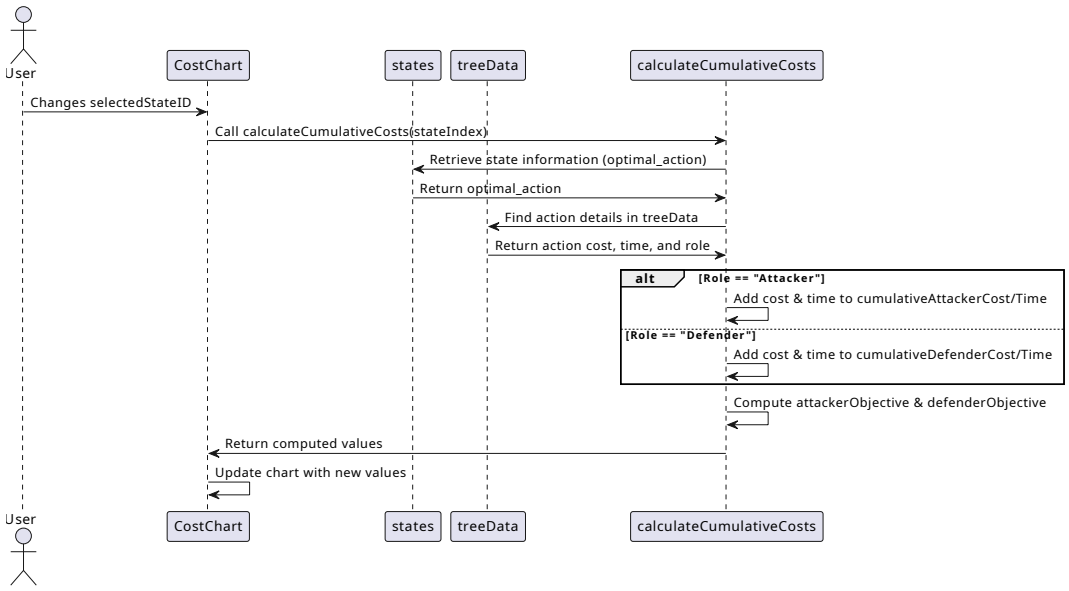

Figure 3.8: Sequence diagram for cumulative cost calculation.

As shown in the diagram, the `CostChart` receives as input the policy states computed by PANACEA and iterates through them to determine the cumulative cost for both the attacker and the defender based on the selected actions. The resulting data is then graphically represented, distinguishing between the already traversed and future parts of the policy.

The `PolicyComparisonChart` component allows users to compare different policies

generated for the same tree, displaying their total monetary and time costs. The data update process involves retrieving information for each available policy in the database, computing the total cost and time required for its execution. The process is illustrated in the following sequence diagram:

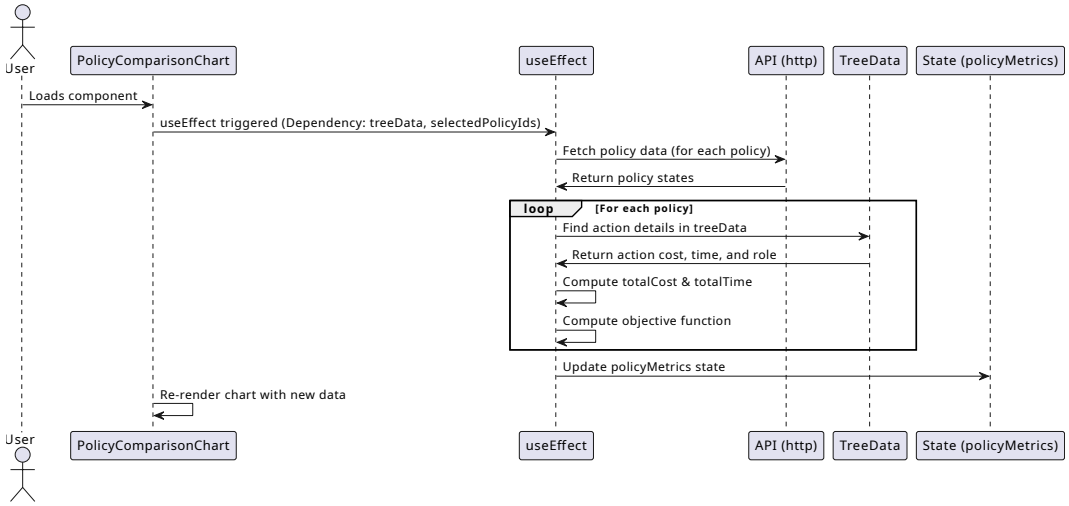

Figure 3.9: Sequence diagram for policy comparison.

As shown in the diagram, when a user selects one or more policies for comparison, the system queries the back end to retrieve the corresponding data. Once received, the key metrics of each policy (cost, time, and objective function value) are computed and displayed in the graph.

Chapter 4

ADTViewer Plugin User Guide

4.1 User Workflow Overview

This chapter provides a structured representation of how users interact with the plugin, from loading an attack tree to analyzing and modifying policies. This workflow is divided into multiple stages, each linked to specific functionalities of the graphical user interface (GUI). By following this process, users can fully utilize the system to explore attack trees, compute optimal policies, and evaluate various attack-defense strategies. The general workflow consists of the following steps:

- Loading an Attack Tree (Section 4.2) Users begin by uploading an XML file containing an attack tree in a format compatible with ADTool. The PANACEA server processes the XML, converting it into a JSON structure suitable for visualization. The attack tree is displayed in the **Tree Visualizer** panel, where users can explore its structure interactively.

- Computing an Optimal Policy (Sections 4.2, 4.4) The PANACEA script processes the attack tree to generate a `.prism` file, which is used to compute an optimal policy using PRISM Games. The resulting policy is returned as a JSON file containing a sequence of states and optimal actions. Users can navigate through these policy states using the **States Visualizer** panel to analyze how the attack-

defense dynamics evolve.

- Modifying the Attack Tree and Recomputing Policies (Section 4.5) Using the **Actions Manager** panel, users can exclude specific attack actions, effectively modifying the attack tree structure. The modified attack tree is exported, and a new policy is computed based on the updated configuration. The **Tree Visualizer** panel is updated, highlighting excluded portions of the tree in gray, while the new policy is loaded into the States Visualizer.

- Analyzing Cost and Comparing Policies (Sections 4.6, 4.7) The **Cost Chart** panel provides a breakdown of the monetary and temporal costs associated with the policy execution. The **Policy Comparison Chart** panel allows users to compare multiple policies, highlighting cost differences and trade-offs between different strategies.

This structured workflow ensures a seamless and efficient user experience, enabling users to explore, modify, and evaluate attack-defense strategies with precision.

4.2 Loading an Attack Tree from an XML File

The **Toolbar** provides various commands for managing the plugin's functionalities, including the option to load an XML file. The process begins by importing an XML file that contains an attack tree in a format compatible with ADTool. This file format is mandatory, as PANACEA's main script processes it to generate an input file suitable for PRISM Games. Figure 4.1 illustrates the **Toolbar** with the XML file loading button.

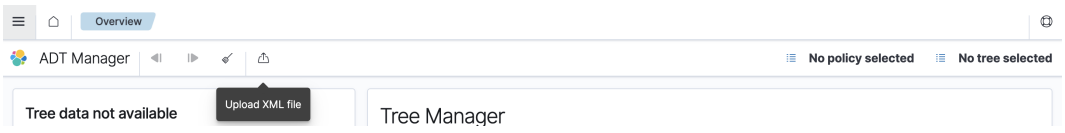

Figure 4.1: **Toolbar** with the XML file loading button.

The XML file follows this structure:

```
<adtree>
    <node refinement="disjunctive">
        <label>Data Exfiltration</label>
        <comment>
            Type: Goal
            Role: Attacker
        </comment>
        <node refinement="disjunctive">
            <label>exfiltrateData</label>
            <comment>Type: Action</comment>
        </node>
    </node>
</adtree>
```

The attack tree structure consists of two main types of nodes:

- **Action Nodes** represent specific steps that an attacker or defender can take, characterized by attributes such as cost, time, and role.
- **Attribute Nodes** define conditions that must be met to achieve a particular goal or perform an action.

Each node has a refinement type, which determines how its child nodes are combined:

- **Disjunctive** (`refinement="disjunctive"`): At least one of the child nodes must be completed to satisfy the parent node.
- **Conjunctive** (`refinement="conjunctive"`): All child nodes must be completed for the parent node to be satisfied.

Once received, the PANACEA server processes the XML file and the first operation is parsing it into a JSON file with the following structure:

```
{
    "nodes": [],
    "edges": []
}
```

This conversion is necessary because, as explained in Section 3.4.3, the visual rendering of the tree is handled by the D3.js library, particularly the `d3.tree()`[1] function, which accepts a list of nodes and edges as input.

The second step involves invoking the main script of PANACEA, which processes the XML file and generates a `.prism` file, passing it directly to PRISM Games. PRISM Games produces a policy as a list of states connected by optimal actions, represented in textual format. The PANACEA server parses this file to generate a JSON representation once again, where each state has a unique ID. Each state is a map where the keys represent node labels, and the values are `0, 1, 2, true, false`:

- `0`, `1` and `2` refer to Attribute nodes, indicating their state in a specific policy state: `0` if inactive, `1` if active, `2` if fixed.
- `true` and `false` refer to Action nodes, with `true` meaning the action is active and `false` meaning it is inactive.

These different node states are visually represented in the tree rendering.

The third step involves the interaction between the PANACEA server and the PostgreSQL database, where the following files are stored:

- The original XML attack tree file is saved in one table.
- The corresponding JSON tree file is stored in another table.
- The JSON policy file is saved in a separate table.
- A mapping table links the XML file to the JSON tree file, ensuring retrieval of the original XML file from any JSON representation.

The structure of the database, including the relationships between the tables, is illustrated in Figure 4.2.

Once stored in the database, the PANACEA server sends a success response to the plugin, containing the newly generated IDs of the policy and tree JSON files. The

[1] `https://d3js.org/d3-hierarchy/tree`

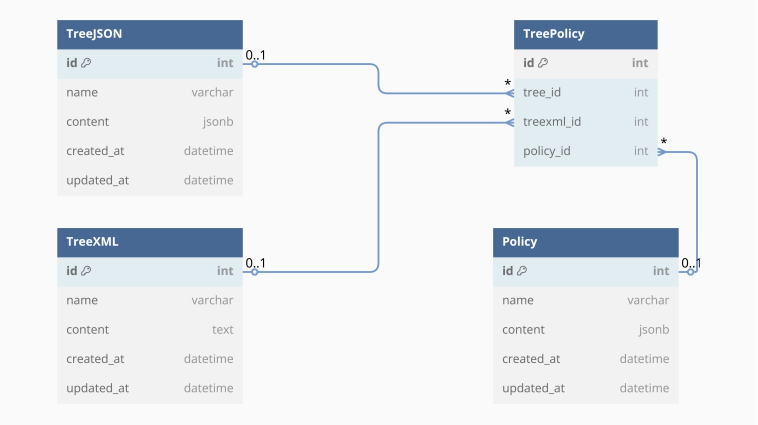

Figure 4.2: Entity-Relationship diagram of the database.

plugin then makes an API call to retrieve and load these files. The sequence diagram in Figure 4.3 illustrates the interaction process.

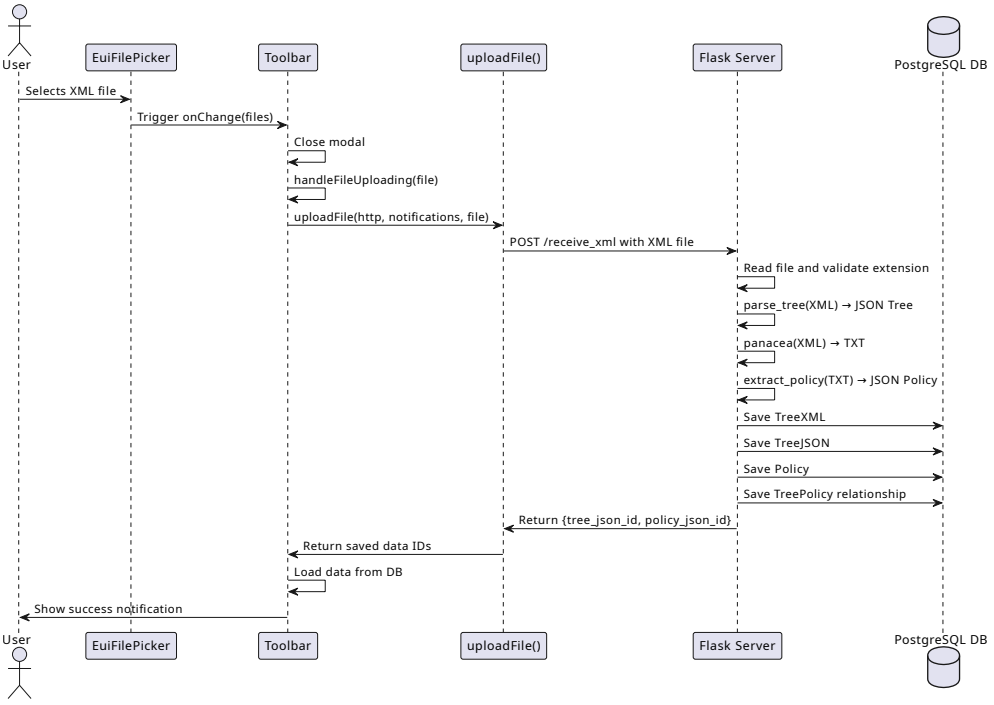

Figure 4.3: Sequence diagram illustrating the interaction between the plugin and the server.

If everything completes successfully, the two files are correctly loaded, and the **Toolbar** updates to display the names of the recently loaded files, as shown by Figure 4.4

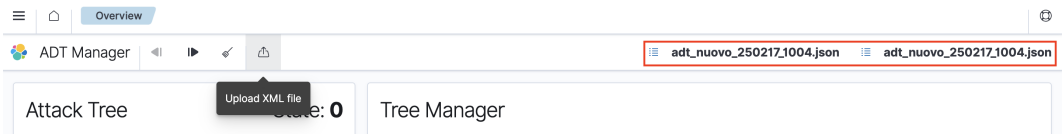

Figure 4.4: Updated **Toolbar** displaying the loaded files.

4.3 Tree Visualizer Panel

The **Tree Visualizer** panel provides a graphical representation of attack trees, allowing users to explore and analyze different attack scenarios interactively. This visualization is helpful in understanding the relationships between different attack steps and the defensive measures that can be taken to mitigate them. Through an intuitive interface, users can navigate the tree structure, select specific nodes, and access detailed information about each attack component.

4.3.1 Attack Tree Representation

The attack tree visualization is implemented using the `d3.tree()` function from the D3.js library. This function constructs a hierarchical tree structure based on the input JSON data derived from the XML attack tree as explained in Section 4.2. Nodes in the tree are represented as ellipses or rectangles, depending on their roles, and edges illustrate the relationships between them. This representation, illustrated in Figure 4.5, is useful to explore the attack tree interactively and enables users to distinguish different types of nodes based on their representation. The visualization supports actions such as dragging, zooming, and panning, allowing users to smoothly explore large attack trees. Nodes can be selected individually or in groups, and selected nodes are visually highlighted to distinguish them from the rest of the tree.

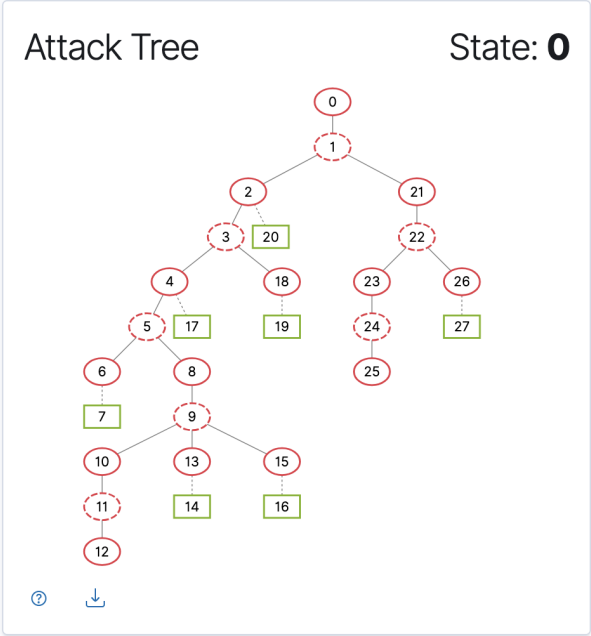

Figure 4.5: Graphical representation of the attack tree using `d3.tree()`. Red ellipses are Action (dashed border) or Attributes (full border) nodes, while green rectangles represent Defender nodes.

4.3.2 Selecting Nodes for Additional Information

The visualization allows users to interact with the tree by clicking on one or more nodes and highlighting them. Selected nodes can be used to retrieve additional information relevant to the attack scenario, such as cost, time, or associated actions. These details are displayed in a separate panel called **Node Info**, which dynamically updates based on the selected nodes. As shown in Figure 4.6, this panel provides a structured overview of node attributes, including its role (Attacker/Defender), type, associated actions, and additional properties. The ability to inspect multiple nodes simultaneously enhances the analytical capabilities of the visualization.

By toggling specific options within the **Node Info** panel, users can choose to display either only the selected nodes or all nodes in the tree. In addition, the panel supports sorting, allowing users to filter the information displayed.

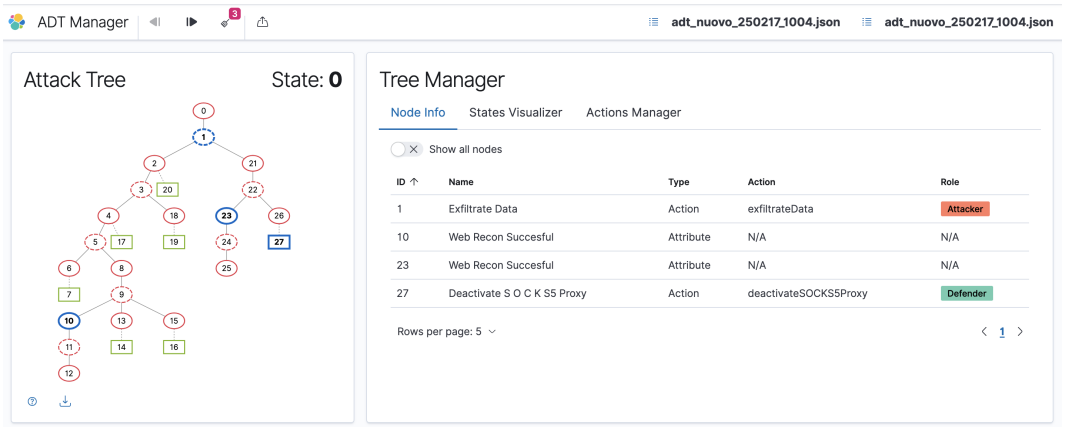

Figure 4.6: **Node Info** panel displaying detailed information about multiple selected nodes.

4.4 States Visualizer Panel

The **States Visualizer** panel allows users to interactively view and explore the states of the policy generated by PANACEA and subsequently loaded through the selector in the **Toolbar**. This panel represents each policy state and illustrates how they are connected through the optimal action, which enables the transition from one state to another. Additionally, it displays information about the cumulative attacker and defender costs associated with each optimal action, which accumulates as the policy progresses.

As depicted in Figure 4.7, the **States Visualizer** panel provides a graphical representation of policy states, allowing users to navigate through different stages of the policy execution. This interaction supports the analysis of the evolution of attack-defense dynamics in conjunction with the **Tree Visualizer** panel, facilitating a deeper understanding of the impact of different actions.

4.4.1 Understanding Policy Representation

As previously mentioned, each policy consists of a set of states, each connected to the next through an optimal action. In the **States Visualizer** panel, states are represented

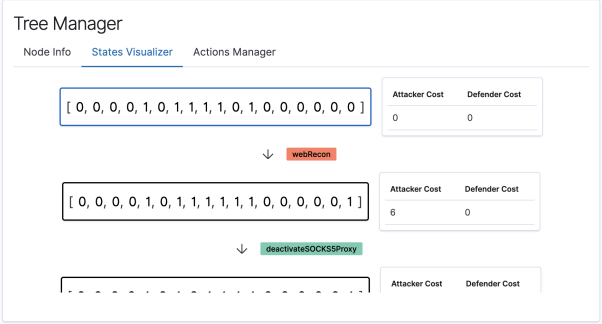

Figure 4.7: The **States Visualizer** panel displaying the graphical representation of policy states and their transitions through optimal actions, along with cumulative attacker and defender costs.

as vectors, where each element corresponds to a unique node label (nodes with the same label are grouped into a single element). The value of each element in the vector represents the state of that node within a specific policy state, as described in Section 4.2. Beside each vector, a tabular summary presents the total attacker and defender costs, computed as the weighted sum of the temporal and monetary costs of all executed actions up to that point.

As shown in Figure 4.8, the state vector is interactive: users can select one or multiple nodes within a vector and see their selection reflected in the graphical representation of the attack tree, and vice versa. This feature enhances usability by allowing users to visually correlate policy states with the structure of the attack tree. Additionally, users can navigate between states either by clicking on a state vector or by using the navigation arrows available in the **Toolbar**. This seamless interaction is ensured by the **Tree Context Provider**, which synchronizes all components, as explained in Section 3.4.2.

4.5 Actions Manager Panel

The **Actions Manager** panel introduces an important feature to the GUI, allowing users to interact directly with PANACEA. Given an existing attack tree and a previ-

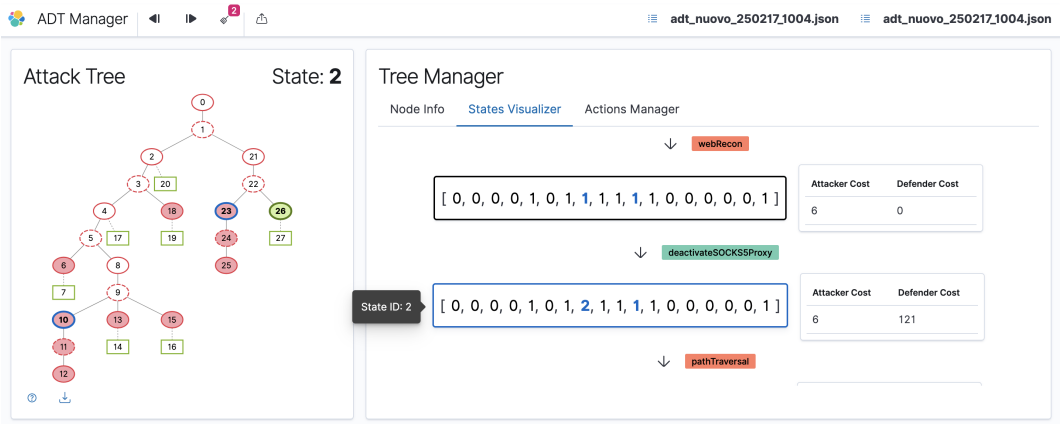

Figure 4.8: Interactive selection of policy states, with real-time mapping to the **Tree Visualizer** panel.

ously computed policy, users may choose to exclude certain actions, effectively removing specific attack paths from the tree. This feature is particularly valuable for simulating various scenarios, such as assuming that certain vulnerable nodes have been "fixed" or analyzing a subset of the available actions. The interface of the **Actions Manager** panel is shown in Figure 4.9, where users can manage actions through an interactive table.

By applying these modifications, PANACEA can compute a new optimal policy and return it to the user. At this point, the user can interact with the newly generated tree using all of the previously discussed functionalities. In addition, new features become available, such as policy comparison, which will be discussed in Section 4.7 through the **Policy Comparison Chart** panel.

4.5.1 Excluding Actions from Policy Computation

The **Actions Manager** panel presents all available actions in a tabular format, equipped with filtering and sorting functionalities. This interactive and dynamic table provides users with two key capabilities:

1. **Selecting an action**: Clicking on a row highlights the corresponding node in

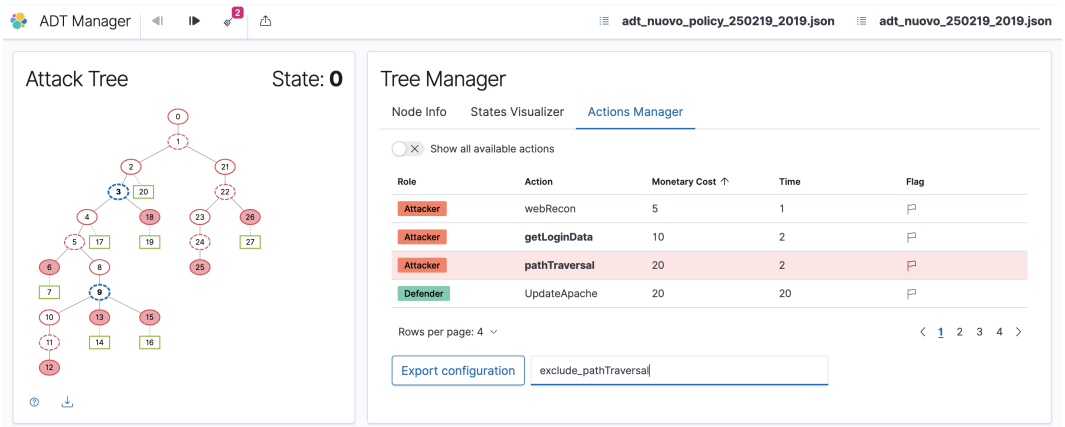

Figure 4.9: Overview of the **Actions Manager** panel, showing the interactive table for managing actions.

the attack tree. Thanks to the **Tree Context**, users can easily establish a visual reference for the selected action.

2. **Flagging an action**: Each row contains a flag icon that, when clicked, marks the action as excluded from policy computation. The flagged row turns red, and during the tree export process, a field `hidden=true` is assigned to all flagged actions.

Multiple actions can be flagged at once, and actions with the same label are automatically flagged together. As explained in Section 3.4.5.1, flagging an action effectively removes the entire subtree rooted at the flagged action(s) from the policy computation.

Once the desired actions are flagged, users can enter a custom name for the new policy. If no name is provided, one is automatically generated. The policy recomputation process is then initiated by clicking the **Export Configuration** button.

A sequence diagram illustrating this interaction in detail is shown in Figure 3.7.

4.5.2 Visualizing the Updated Attack Tree and Policy

After the new policy computation is complete, a new attack tree and a new policy are generated. The updated attack tree maintains the same structure as the original

one, but excluded subtrees are now displayed in gray. Users can still interact with these grayed-out nodes to retrieve information, but they are no longer considered in the policy computation. This visual distinction allows users to immediately recognize which parts of the attack tree are being considered and which have been excluded.

Similarly, the new policy reflects these modifications. The **States Visualizer** panel updates the state vectors, which now contain fewer elements, corresponding to the remaining nodes. Furthermore, the optimal actions between states may differ from those in the original policy, as there are less available actions now.

Figure 4.10 illustrates an example of the updated attack tree and its corresponding policy representation after excluding actions.

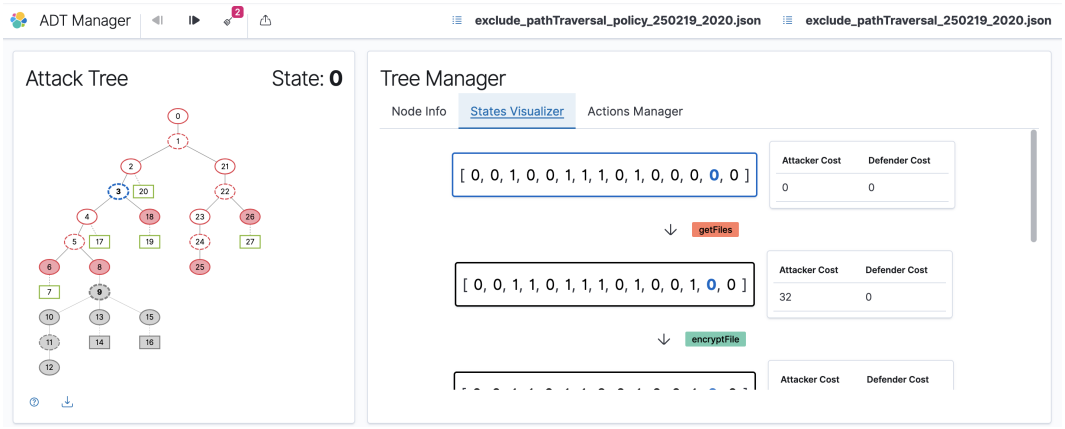

Figure 4.10: Updated attack tree (left) and modified policy representation (right) after excluding actions.

This functionality provides users with an effective way to analyze alternative attack-defense strategies, simulate different security measures, and gain deeper insights into the dynamics of attack trees and their associated policies.

4.6 Cost Chart Panel

The **Cost Chart** panel allows users to visualize the monetary and temporal cost evolution throughout the different states of a computed policy. This panel presents a dynamic

line chart, where the x-axis represents the policy states, and the y-axis represents the objective function value, defined as the weighted sum of monetary and temporal costs.

This visualization provides users with both a general overview of the policy cost and the ability to identify costly state transitions, where a single action significantly increases the total cost. The graph includes three key curves:

- The **attacker objective function**, which quantifies the cumulative cost for the attacker and is displayed in red.
- The **defender objective function**, representing the cumulative defender cost and shown in blue.
- The **total objective function**, which combines both attacker and defender costs, plotted in purple.

4.6.1 Visualizing Cost Evolution Across Policy States

The **Cost Chart** panel provides an interactive visualization that dynamically updates as users navigate through policy states. The current state can be changed using the navigation controls in the **Toolbar**, by clicking on a vector in the **States Visualizer** panel, or by directly interacting with the **Cost Chart** panel itself. When the selected state changes, the chart adjusts accordingly, highlighting the cost values up to the current state while keeping future states visually distinct.

As illustrated in Figure 4.11, the chart displays three curves representing the attacker, defender, and total objective functions. The attacker's cost function is displayed in red, with a semi-transparent red area shading past states, while the defender's cost function is shown in blue, with a similar blue shading for past states. The total cost function is plotted in purple, summarizing the combined cost of both attacker and defender. Future states that have not yet been reached remain gray, providing a clear distinction between executed and upcoming actions.

Users can hover over the lines in the chart to obtain detailed information about the current state and the corresponding values of the objective functions. This interactive

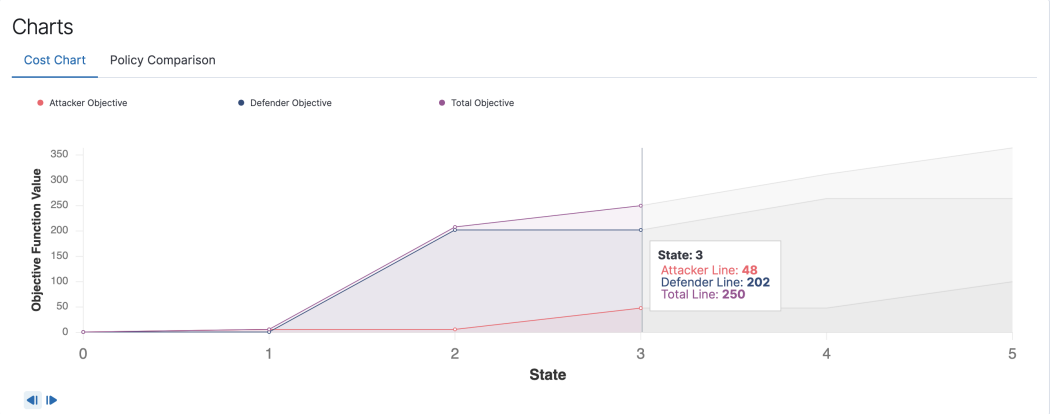

Figure 4.11: Cost evolution curves for the attacker (red), defender (blue), and total cost (purple). The shaded areas highlight past states, while future states remain in gray. The tooltip displays the objective function values at the selected state.

tooltip displays the state number and the exact cost values for the attacker, defender, and total objective functions up to that state, as shown in Figure 4.11. This feature helps users quickly analyze cost trends and detect critical transitions, where a specific action causes a significant increase in cost.

4.6.2 Cost Breakdown: Attacker vs. Defender

The **Cost Chart** panel also enables users to analyze how the overall cost is distributed between the attacker and the defender. In an attack-defense model, these two entities incur different costs depending on their roles in the system. The attacker's costs include financial expenses, execution time, and resource investments required to carry out an attack, while the defender's costs represent the effort needed to mitigate or prevent attacks, such as deploying countermeasures or reinforcing system defenses.

As illustrated in Figure 4.11, the chart provides a clear view of how each policy state contributes to the cumulative cost for both parties.

Additionally, the objective function can be adjusted to assign different weights to monetary cost and execution time, allowing users to explore alternative scenarios where either factor is prioritized.

4.7 Policy Comparison Chart Panel

The **Policy Comparison Chart** panel allows users to analyze multiple policies simultaneously, providing a visual representation of their trade-offs in terms of monetary cost and execution time. This feature is particularly useful for evaluating different attack-defense strategies and selecting the most effective response to a given attack scenario.

4.7.1 Comparing Multiple Policies

The panel presents a line chart where each policy is plotted based on its execution time (x-axis) and monetary cost (y-axis). Each line represents a distinct policy, allowing users to visually compare how costs evolve over time for different strategies.

As shown in Figure 4.12, users can select multiple policies from a list and display them on the same graph. Additionally, users can interact with the graph by hovering over specific points to retrieve precise information about each policy's metrics at a given time step. The tooltip provides details such as the policy name, total time required, monetary cost, and the computed objective function value. This feature enhances the comparison process by allowing for a more granular evaluation of the differences between policies.

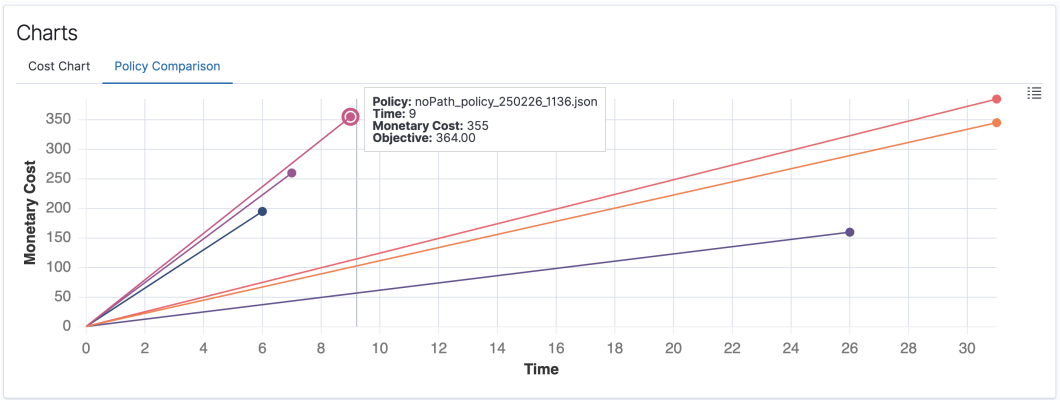

Figure 4.12: Comparison of multiple policies, illustrating their trade-offs between time and monetary cost. The tooltip provides detailed information about a selected policy.

Chapter 5

Case Study

This chapter presents a case study in which we analyze two different attack-defense scenarios using the ADTViewer plugin and the PANACEA framework. All visualizations presented in this chapter are generated using the plugin itself. In the first scenario, we consider an unrestricted attack path, following the optimal policy computed by the framework. In the second scenario, we mitigate the attack by flagging and removing the action **pathTraversal**. By comparing the two cases, we can evaluate the impact of mitigation strategies on the attack path and cost distribution.

5.1 Attack Scenario 1: Unrestricted Attack Path

In this scenario, the attack path is executed without any restrictions, meaning all available attack and defense actions are permitted. The root node of the attack tree represents the attacker's final objective which is data exfiltration, supported by various attack steps such as *reverse shell*, *file access*, and *buffer overflow* exploits. There are also available several defense actions such as *Apache reconfiguration*, *file encryption*, and *credentials change*. Figure 5.1 illustrates the initial structure of the tree.

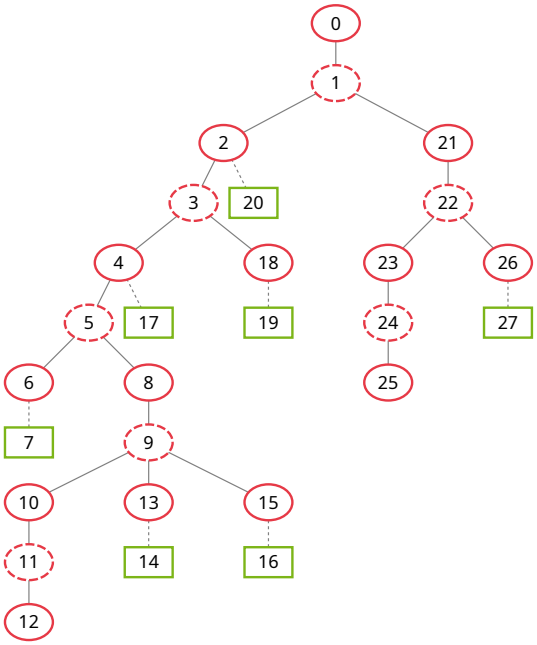

Figure 5.1: Attack defense tree structure. Red nodes with full border are Attribute nodes, red nodes with dashed border are Action nodes. Green nodes represent defender actions.

5.1.1 Description of Attack Path

The optimal attack sequence computed by PANACEA unfolds as follows. The attack begins with **webRecon**, a malicious action that aims to collect information on the web server. With this newly acquired information, the attacker proceeds, but at this stage, the defender intervenes by executing **deactivateSOCKS5Proxy**, a defensive measure designed to disable a proxy service that could facilitate malicious access. After that, the attacker exploits a **pathTraversal** vulnerability, which grants unauthorized access to restricted directories. In response, the defender takes a countermeasure by executing **reconfigureApache**, an action intended to mitigate misconfigurations in the Apache server and reinforce its security.

Figure 5.2 illustrates the evolution of the attack path according to the optimal policy computed by PANACEA, showing the sequence of actions taken by the attacker and the associated costs for both the attacker and the defender.

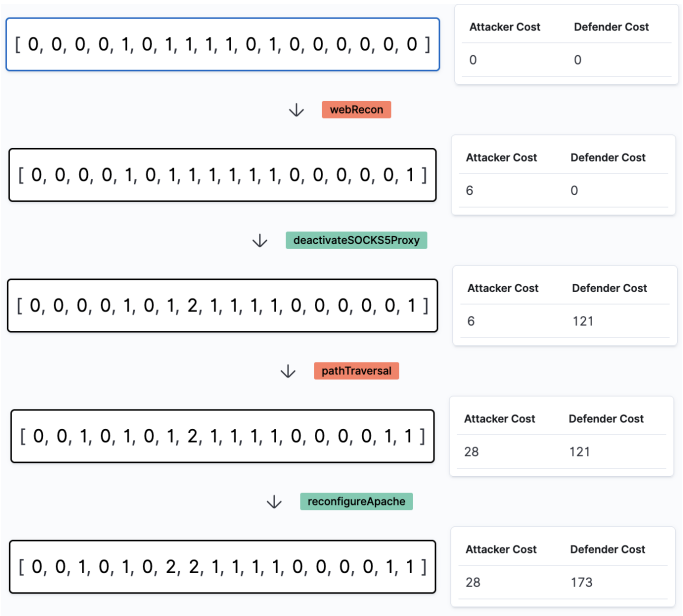

Figure 5.2: Attack path evolution following the optimal policy, highlighting attacker and defender costs at each step.

5.1.2 Cost and Impact Analysis

The graph illustrated in Figure 5.3 shows how costs accumulate across the different policy states, highlighting the increasing investment required to progress through the attack path, as well as the defensive mitigation expenses incurred by the system.

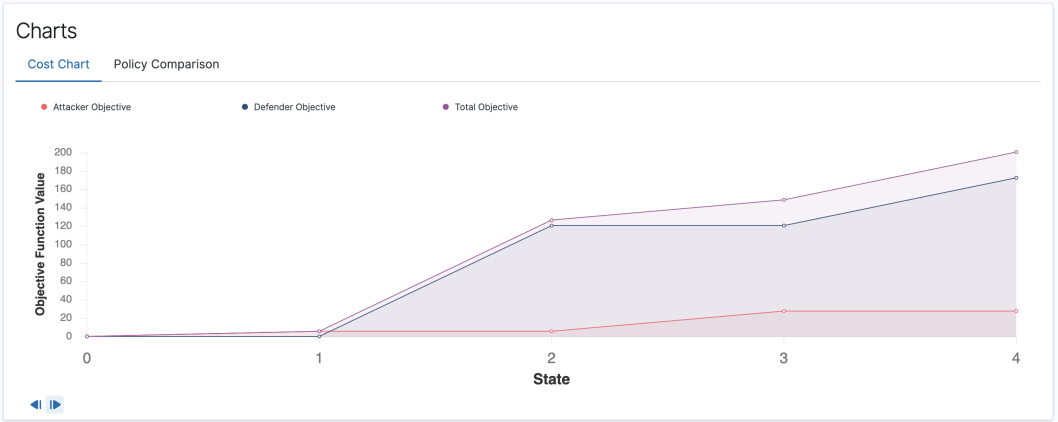

Figure 5.3: Cost dynamics between the attacker and defender in Scenario 1.

As shown, the introduction of defense actions like **deactivateSOCKS5Proxy** and **reconfigureApache** adds cost to the defender, but also affects the efficiency of the attacker.

Beyond financial implications, the potential impact of this attack scenario could extend across multiple aspects of an organization. Unauthorized access to sensitive files might result in severe data breaches, potentially leading to leaks and privacy violations. These security failures could carry significant financial repercussions, as organizations may face costs associated with incident response, regulatory fines, and even potential ransom payments. Additionally, operational disruptions caused by the compromise of critical systems might lead to service downtime, reducing productivity and affecting business continuity.

5.2 Attack Scenario 2: Mitigated Attack Path

In this scenario, we introduce a mitigation strategy by flagging and removing the **pathTraversal** attack action. As shown in Figure 5.4, the modified attack tree retains the overall structure of the original tree illustrated in Figure 5.1. However, the node and subtree associated with the **pathTraversal** action is now grayed out, indicating that it is no longer accessible in potential attack paths. This mitigation directly impacts the attacker's ability to exploit this vulnerability, forcing them to pursue alternative, potentially costlier, or less effective strategies.

5.2.1 Description of Attack Path

The updated optimal attack sequence computed by PANACEA unfolds as follows. As in the unrestricted scenario, the attack begins with **webRecon**, allowing the attacker to collect information on the target system. However, in this mitigated scenario, the defender takes proactive measures by executing **changeCredentials**, thereby reinforcing authentication mechanisms. In response, the attacker adapts their approach and attempts a **bufferOverflow** attack, aiming to escalate privileges and gain execution con-

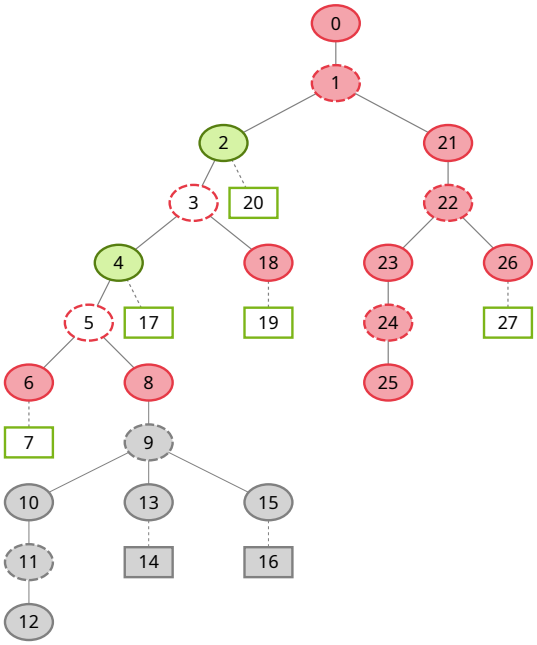

Figure 5.4: Attack defense tree representation after flagging the **pathTraversal** action. The node associated with the **pathTraversal** action, along with its subtree, is grayed out, indicating it can no longer be part of attack paths.

trol over the system. The defender reacts by implementing **changeFilePermissions**, which restricts access to sensitive files in order to minimize potential damage. Despite these defensive interventions, the attacker ultimately proceeds with **exfiltrateData**, achieving data extraction, although with increased difficulty and associated costs.

By eliminating the **pathTraversal** action, the structure of the attack path is altered, necessitating alternative exploitation methods that influence both operational efficiency and resource consumption.

5.2.2 Cost and Impact Analysis

To further assess the implications of the mitigation strategy, we analyze the cost evolution for the mitigated attack path. Figure 5.5 presents the cost distribution between the attacker and the defender throughout the execution of the revised policy.

The mitigated attack path introduces increased costs for the attacker, requiring additional steps to achieve their objective. Similarly, the defender incurs a higher cost

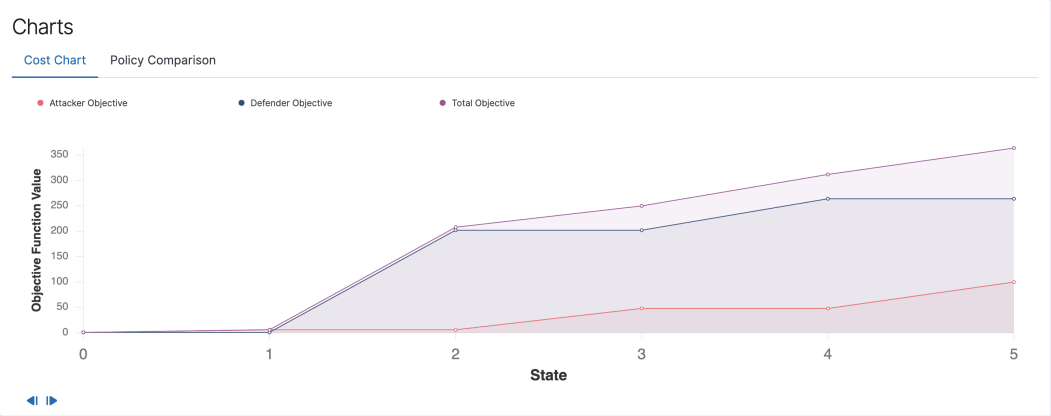

Figure 5.5: Cost evolution chart for the mitigated attack path.

due to proactive security measures such as **changeCredentials** and **changeFilePermissions**. These defensive actions contribute to the overall cost of protection, but effectively alter the attack dynamics, making it more difficult and resource-intensive for the attacker to succeed.

5.3 Comparative Analysis of Attack Scenarios

To evaluate the impact of the mitigation strategy, we compare the evolution of the cost in both scenarios. Figure 5.6 provides a comparison of the policy costs in the unrestricted and mitigated cases.

From the graph, we observe several key differences between the two scenarios:

- **Increased Attacker Cost**: The mitigated scenario forces the attacker to take alternative, often more complex, steps to achieve the attack goal. This results in higher cumulative costs and extended attack duration, as the attacker needs to find and exploit different vulnerabilities.
- **Higher Defender Investment**: The defender incurs additional costs in the mitigated case due to proactive countermeasures such as **changeCredentials** and **changeFilePermissions**.

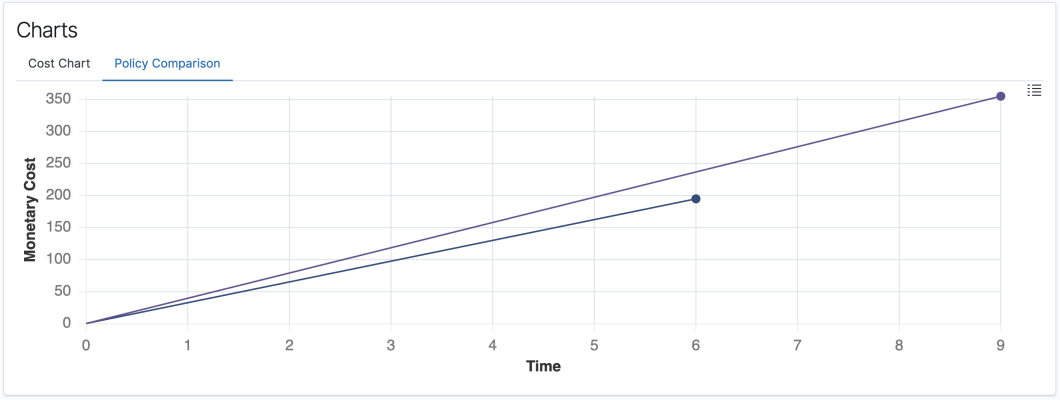

Figure 5.6: Comparison of policy costs for unrestricted (blue line) and mitigated (purple line) attack paths.

- **Impact on Attack Efficiency**:In the unrestricted scenario, the attack follows a more direct and cost-efficient route. In contrast, the mitigated attack path introduces additional barriers, making it less efficient for the attacker. This change suggests that even partial mitigations can disrupt and complicate an adversary's plan, potentially discouraging certain attack paths.

Beyond the cost dynamics, it is important to consider the broader implications of these findings. By analyzing how mitigations affect the overall attack path, security teams can better allocate resources to counter the most impactful threats. Furthermore, the results suggest that combining multiple defensive actions can lead to a more resilient system, even if the attack goal remains theoretically achievable.

Chapter 6

Conclusions and Future Developments

This thesis presented the design and implementation of a Graphical User Interface for an Intrusion Response System, focusing on attack tree analysis and the integration of game-theoretic security modeling. The developed system, implemented as an OpenSearch Dashboards plugin, provides an intuitive visualization of Attack Defense Trees, enabling security analysts to better understand, modify, and evaluate different attack-defense scenarios. Using the PANACEA framework in conjunction with PRISM-Games, the solution facilitates decision-making processes by computing optimal defense policies and presenting them within an interactive environment.

6.1 Summary of Findings

This thesis has introduced a structured approach to attack tree analysis, addressing limitations in existing solutions through the following contributions:

1. **Design and Implementation of a Modular GUI**: The ADTViewer plugin provides a user-friendly interface, enabling users to interact with attack defense trees without relying on the use of command-line interfaces or formal verification methods.

2. **Integration with OpenSearch Dashboards**: The plugin's integration with OpenSearch Dashboards allows seamless interoperability with Security Information and Event Management (SIEM) solutions such as Wazuh, providing users a more comprehensive threat monitoring and analysis environment.

3. **Automated Attack Policy Computation**: The integration with PRISM-Games allows users to generate optimal attack-defense strategies, providing a systematic way to assess security risks and mitigation measures.

4. **Scenario Analysis and Policy Comparisons**: The plugin facilitates the evaluation of alternative defensive strategies by allowing users to modify attack trees and compare different security policies, supporting informed decision-making.

6.2 Limitations and Challenges

Despite its advantages, the current implementation presents some limitations that should be addressed:

1. **Computational Overhead**: The process of generating optimal policies can be resource-intensive, particularly for large and complex attack trees.

2. **Dependency on External Tools**: The reliance on PRISM-Games for policy computation requires users to have access to specific verification frameworks, which may pose challenges for deployment in restricted environments.

3. **User Experience Improvements**: While the interface is designed for usability, further refinements in navigation and interactivity could enhance the overall user experience.

6.3 Suggestions for Future Development

To further advance this research and improve the effectiveness of the tool, several future developments are proposed:

1. **Performance Optimization**: Implementing caching mechanisms and optimizing API calls can reduce computation time and improve responsiveness.

2. **Extended Support for Defensive Measures**: Enhancing the tool to model and visualize more complex defensive strategies, including adaptive responses and real-time countermeasures, would broaden its applicability.

3. **Integration with Machine Learning Techniques**: Leveraging AI-driven anomaly detection and risk assessment models could provide automated recommendations for security policies.

4. **User Feedback and Iterative Improvements**: Conducting user studies and gathering feedback from cybersecurity specialists would help refine the interface and functionalities to better meet practical needs.

In conclusion, this thesis has demonstrated the feasibility and benefits of a GUI-based approach to attack tree analysis, effectively bridging the gap between theoretical security models and practical applications. By making advanced cybersecurity modeling tools more accessible, this research contributes to improving the efficiency of security decision-making and response planning. Continued development and integration with emerging technologies will further enhance its impact, enabling organizations to stay ahead of evolving cyber threats.

Bibliography

[AIA21] Mohammad Aljanabi, Mohd Arfian Ismail, and Ahmed Hussein Ali. Intrusion detection systems, issues, challenges, and needs. *International Journal of Computational Intelligence Systems*, 14:560–571, 2021.

[AMZZ+17] Shahid Anwar, Jasni Mohamad Zain, Mohamad Fadli Zolkipli, Zakira Inayat, Suleman Khan, Bokolo Anthony, and Victor Chang. From intrusion detection to an intrusion response system: Fundamentals, requirements, and future directions. *Algorithms*, 10(2), 2017.

[BCG18] James Braunstein, Edward Cartwright, and Andreas Gutjahr. Risk management using attack trees and game theory. *Computers & Security*, 77:81–91, 2018.

[BFP06] S. Bistarelli, F. Fioravanti, and P. Peretti. Defense trees for economic evaluation of security investments. In *First International Conference on Availability, Reliability and Security (ARES'06)*, pages 8 pp.–423, 2006.

[FKN+20] Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman, David Parker, and James Worrell. Verification and strategy synthesis for stochastic games with multiple objectives. *Journal of Automated Reasoning*, 64(4):795–831, 2020.

[GHL+16] Olga Gadyatskaya, René Rydhof Hansen, Kim Guldstrand Larsen, Axel Legay, Mads Chr Olesen, and Danny Bøgsted Poulsen. Modelling attack-defense trees using timed automata. In *FORMATS 2016 - 14th Interna-*

tional Conference on Formal Modelling and Analysis of Timed Systems, pages 35–50. Springer, August 2016.

[GKM23] Afrah Gueriani, Hamza Kheddar, and Ahmed Cherif Mazari. Deep reinforcement learning for intrusion detection in iot: A survey. In *2023 2nd International Conference on Electronics, Energy and Measurement (IC2EM)*, volume 1, pages 1–7, 2023.

[II89] John Q. Walker II. A node-positioning algorithm for general trees. *Software: Practice and Experience*, 20(7):685–705, 1989.

[KKMS13] Barbara Kordy, Piotr Kordy, Sjouke Mauw, and Patrick Schweitzer. Adtool: Security analysis with attack–defense trees. In Kaustubh Joshi, Markus Siegle, Mariëlle Stoelinga, and Pedro R. D'Argenio, editors, *Quantitative Evaluation of Systems*, pages 173–176, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[KMRS12] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. Attack–defense trees. *Journal of Logic and Computation*, 24(1):55–87, June 2012.

[KMRS14] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. Attack–defense trees. *Journal of Logic and Computation*, 24(1):55–87, 2014.

[KNP11] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification*, pages 585–591, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[Mar23] Valerio Marini. Panacea: Attack tree analysis with prism-games, 2023. Accessed: 2024-02-06.

[NFW17] Vidhyashree Nagaraju, Lance Fiondella, and Thierry Wandji. A survey of fault and attack tree modeling and analysis for cyber risk management. In *2017 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–6, 2017.

[PR19] Oskars Podzins and Andrejs Romanovs. Why siem is irreplaceable in a secure it environment? In *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*, pages 1–5, 2019.

[Pro22] OpenSearch Project. Introduction to opensearch dashboards plugins, 2022. Accessed: 2024-02-06.

[RRI+20] Erkuden Rios, Angel Rego, Eider Iturbe, Marivi Higuero, and Xabier Larrucea. Continuous quantitative risk management in smart grids using attack defense trees. *Sensors*, 20(16):1–18, 2020.

[Sch99] Bruce Schneier. Attack trees. *Schneier on Security*, 1999. Accessed: 2024-02-06.

[SSEJJD12] Alireza Shameli-Sendi, Naser Ezzati-Jivan, Masoume Jabbarifar, and Michel Dagenais. Intrusion response systems: Survey and taxonomy. *International Journal of Computer Science and Network Security (IJCSNS)*, 12, January 2012.

[SSH+18] Muhammad Ali Siddiqi, Robert M. Seepers, Mohammad Hamad, Vassilis Prevelakis, and Christos Strydis. Attack-tree-based threat modeling of medical implants. In *International Workshop on Security Proofs for Embedded Systems*, 2018.

[SSSW98] C. Salter, O. S. Saydjari, B. Schneier, and J. Wallner. Toward a secure system engineering methodology. In *Proceedings of the 1998 Workshop on New Security Paradigms (NSPW '98)*, pages 2–10, 1998.

[TMA+22] Aamna Tariq, Jawad Manzoor, Muhammad Ammar Aziz, Zain Ul Abideen Tariq, and Ammar Masood. Open source siem solutions for an enterprise. *Information & Computer Security*, 31(1):88–107, 2022.

[TRP+24] Cheryl Ching Tay, Darshan Kumar S/O Ramesh, Rachel Ying Xuan Poh, Randy Yi Ping Cheang, Villarico Gabriel Babiera, Huaqun Guo, and Liming Lu. The feasibility of opensearch's security analytics with wazuh. 9 2024.

[Wei91] Jonathan D Weiss. A system security engineering process. In *Proceedings of the 14th National Computer Security Conference*, volume 249, pages 572–581, 1991.

[XAH+16] Chao Xu, Man Ho Au, Xinyi Huang, Joseph K. Liu, and Zhenfu Cao. Security and privacy in smart home: A survey. *IEEE Communications Surveys & Tutorials*, 18(2):1354–1376, 2016.