Diploma Thesis

A Configurable Application Suite for Streamlined Anomaly Detection Algorithm Development

Panagiotis Papadopoulos

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

May 2023

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

A Configurable Application Suite for Streamlined Anomaly Detection Algorithm Development

Panagiotis Papadopoulos

Supervisor professor:
Vasos Vassiliou

The final project was submitted in partial fulfilment of the requirements for obtaining the Computer Science degree of the Department of Computer Science of the University of Cyprus

May 2023

Acknowledgments

I would like to express my deepest gratitude to Professor Vasos Vassiliou, my thesis supervisor, for his continuous support throughout the course of this project. His profound knowledge, insightful critiques, and patient guidance have been invaluable to my work.

His enthusiasm for the subject and commitment to advancing my understanding significantly influenced my academic journey. The door to Professor Vassiliou’s office was always open whenever I ran into a roadblock or had a question about my research or writing.

Furthermore, I would like to thank the rest of the Computer Science faculty and staff for their encouragement and assistance over the course of my studies. My thanks also go out to my peers for their companionship and constructive feedback during the development of this project.

Lastly, I would like to express my heartfelt appreciation to my family and friends for their unwavering belief in my abilities and their ongoing support throughout my academic career. This accomplishment would not have been possible without the collective support and guidance of these individuals.

Abstract

This thesis presents the development and implementation of a real-time, configurable testing environment aimed at streamlining the development of detection algorithms in Intrusion Detection Systems (IDS). In the constantly evolving field of Network Security, there is an increasing need for more effective and efficient threat detection systems. However, the development of new and improved algorithms can be a difficult and time-consuming process.

This project aims to address this issue by providing a suite of applications that simulate the functionalities of an IDS within a computer network environment. It allows for real-time data capturing, processing, storage, and result presentation, providing a realistic environment for algorithm testing. A unique feature of this tool is its configurable threat detection module, which can be adapted to include and use different anomaly detection algorithms, enabling multiple algorithms to be tested concurrently.

Ultimately, the goal of this tool is to speed up the development process by providing a flexible and realistic testing framework, thereby facilitating the introduction of more effective threat detection systems into real-world applications.

Content

Chapter 1

Introduction

1.1 Introduction to IDS

An Intrusion Detection System (IDS) is a system designed to detect potential threats in a computer network. To accomplish this task multiple components are used and communicate together. The process begins with the capture of network traffic, this network traffic represents raw data. Raw data is then processed to extract information that can be used to detect anomalies. Finally, processed data is used as input into algorithms that are designed to detect abnormalities. The output of this algorithm is an estimation of the state in which the network is in according to the presented data. An IDS could perform other actions such as monitoring and can even take actions depending on the detected state of the network. However, threat detection is the core functionality of any IDS. These systems are employed by network administrators as an additional layer of security to their network which can react and issue warnings the moment an abnormality is detected. [4]

1.2 Aim of the final project

The aim of this project is to develop a suite of applications that can be used as a testing environment for new approaches in anomaly detection algorithms. This testing environment should emulate the functions of an IDS in a computer network which includes data capturing, data processing, data storage and an interface where the results are presented. Finally, the application suite should include a configurable module that can be used to change the detection algorithms that are used to make predictions.

The use of such a tool can help in the development of new or improved algorithms for anomaly detection. This can be achieved by providing a framework where new detection algorithms can be tested in a realistic and real-time environment. In addition, this application suite can allow for multiple algorithms to be tested at the same time which can further increase the speed of development.

1.3 Structure of the final project

The first chapter of this document serves as an introduction for the project. It includes an abstract description of the project and a brief description of an Intrusion Detection System (IDS). Some concepts that are mentioned in the description of the IDS and the project are explained in detail in a later chapter. The structure of the document is also included in the first section.

The second chapter contains sections which describe the need that led to the development of this project and from where that need originated. Furthermore, other similar existing projects are briefly mentioned and discussed.

The third chapter consists of sections where concepts essential to the project are discussed along with the technologies that have been leveraged to develop this application suite such as programming languages and frameworks. These concepts and technologies include but are not limited to Computer Networks, Network Protocols, Artificial Intelligence, Machine Learning, and .NET Core.

The fourth chapter discusses the requirements of this project. The functional and non-functional requirements are mentioned in detail along with the project's specifications. Use Case Diagrams are also included. This chapter will also focus on who are expected to make use of the application suite.

The fifth chapter contains sections that describe the design and implementation of the project. Decisions about architecture will be discussed in depth and all components will be explained in detail.

The sixth chapter contains the testing and results of this project. The methodology used to get these results will also be defined here. Finally, a section with the conclusions that we have arrived at will be contained in the chapter.

The last and seventh chapter discusses possible future work that can be done after this project.

References can be found in the appendices section at the end of the document.

Chapter 2

Problem/Similar Software

2.1 Problem-Need

Since the goal of an IDS is to detect anomalies in network traffic and the features that can be extracted from a given dataset of network traffic are fixed, the accuracy of the predictions depends solely on the detection algorithm used. Multiple detection algorithms exist and have been implemented over time with varying degrees of success. Success in this context represents the accuracy of the IDS predictions, where an accurate prediction means that the IDS has correctly detected whether the network is under-attack/malfunction or that operation is normal.

Each algorithm that is used for detection follows a principle to reach a prediction. These approaches can be Conditional, where predictions are made depending on a collection of conditions that need to be met, these conditions can be defined based on previous attacks and their signature such as a very specific set of actions. Another approach is Statistical Analysis, where predictions are made depending on the deviation that exists between what is currently detected and what is considered normal behavior. A similar approach to Statistical Analysis is Time Series Analysis where the changes in network traffic patterns are observed over time and checked for any sudden or unusual changes. [5]

An IDS can make use of multiple algorithms each with a different approach to reach a final prediction however, predictions need to be made quickly since data is collected on a real-time basis and the window of time between consecutive predictions ranges from a few minutes down to a few milliseconds. This limitation presents an issue with conventional detection algorithm approaches.

The issue is that the processing power they need to produce results scales up when the network scales up in size and as a result an increase in raw data that is generated. Also, as more conditions, analysis and comparisons need to be performed to increase the accuracy of these algorithms, there is a further increase is processing power needs. Considering that the hardware on which an IDS runs has a limited amount of processing power that it can produce, and hardware upgrades come at a cost, there is a clear incentive for new approaches with better scaling to be introduced.

2.2 Description of potential solutions

Artificial Intelligence and Machine Learning have seen a rise in recent years and have provided solutions in other sectors such as Image and Voice Recognition. In these approaches a model is constructed and using large quantities of labelled data are trained to then be able to make accurate predictions on data that it has never seen before. This approach appears to be like Statistical Analysis, but here the patterns analyzed are not defined by the developer but instead the model during its training.

Multiple models exist and each model can be trained using a variety of training algorithms. Furthermore, these models are not well-defined structures but rather abstract definitions which allow for experimentation. This experimentation is crucial as each model and training algorithm has its own strengths and weaknesses, in addition, the structure of the model needs to be fine-tuned for optimal results.

Despite the difficulty in implementing such approaches, research has shown that if implemented correctly, Machine Learning approaches can have increased accuracy compared to other approaches. However, it is important to understand that a Machine Learning model can have varying degrees of success in different networks and therefore every implementation should be tested and tuned for optimal results. This testing and tuning of the model can be time consuming, this is because the training and testing of these models can be a lengthy process especially for more complex models. Furthermore, there is no easy way to test multiple models at once. Usually, comparisons between models are done using pre-existing datasets, these datasets could contain vast amounts of data but are not always a good representation of the data that the target network is producing and therefore could be a source of error in the results.

This is where a need for a suite of tools which can help speed up the process of training, testing, and comparing of possible models arises. A suite of tools that can allow for the testing of multiple models at once with real-time data and an interface that can display the results of these models can potentially accelerate the implementation of Machine Learning models.

2.3 Similar Software

2.3.1 Snort

Snort is an Open-Source Intrusion Prevention System developed by the company Sourcefire which was bought by Cisco Systems Inc in 2013. It allows for real-time traffic analysis, logging of incoming traffic and intrusion detection. To detect and react to possible threats the user must define a set of rules and actions. Rulesets can be created by the user, but they can also access the publicly available Rulesets of the community. Another option is to subscribe to the Ruleset that is maintained and updated by Cisco Systems. This Ruleset is a superset of the community Ruleset, however subscription to this Ruleset comes at a cost.

While Snort does offer some flexibility for defining rules and actions, the user is limited to a Conditional approach. Furthermore, Snort does not allow for the comparative testing of multiple detection models. [16]

Figure 1: Snort logo

2.3.2 Suricata

Suricata is an Open-Source software that can serve multiple functions, but the primary function is an Intrusion Detection System and Intrusion Prevention System. It offers a vast collection of tools that can assist a network administrator in the monitoring of the network. The primary approach for detecting threats is performing traffic analysis to detect signatures of known threats. In addition, a Conditional approach is also used with the Suricata Ruleset and VRT Ruleset which are updated regularly. Finally, Suricata allows the user to develop and deploy Lua scripts that can be used for threat detection.

Overall, Suricata offers flexibility and customizability, however the user is limited to Conditional and Statistical Analysis approaches. The functionality of Lua Scripting does give some room for different approaches, but developing and implementing complex models this way is not recommended. [18]

Figure 2: Suricata logo

2.3.3 Zeek

Zeek is an Open-Source software for traffic analysis in computer networks. It is primarily used as a monitoring tool thanks to its comprehensive library of tools that are tailored towards this task. It can allow for extensive logging of network activity in all the layers of the network, from physical wire connections to high level application protocols such as HTTP. Furthermore, Zeek is equipped with tools that can assist in analysis of traffic and the detection of threats. Zeek also allows for a lot of flexibility and customization thanks to its custom-made scripting language for traffic analysis. Another crucial aspect of Zeek is its capability to run on commodity hardware that is more affordable compared to proprietary solutions.

All in all, Zeek offers an extensive, flexible, and customizable library of tools to the user, however the primary objective of Zeek is monitoring and traffic analysis and not threat detection. This means that Zeek is better utilized as a tool for extracting data which can later be used as input to an IDS. [17]

Figure 3: Zeek logo

Chapter 3

Required Knowledge and Technologies

3.1 Basic Definitions

3.1.1 Artificial Intelligence

According to John McCarthy, Artificial Intelligence refers to the science of engineering and developing intelligent machines, systems, and programs. It has similarities with another field of study, where computers and machines are used to understand human intelligence, but Artificial Intelligence is not limited to biologically observable methods. [1]

The field of Artificial Intelligence is extensive and contains other concepts such as Machine Learning which will be discussed later in the document. Artificial Intelligence also has a long history dating back to the 1950s and throughout time has had periods of growth and stagnation.

Artificial Intelligence systems can perform their tasks by grasping their environment and use problem solving to achieve a goal. Their problem-solving capabilities allow them to solve problems that traditional algorithmic approaches cannot solve or require a lot of processing power. Furthermore, their actions can be influenced by previous experience to further improve future results.

3.1.2 Machine Learning

Machine Learning is a form of Artificial Intelligence. Machine learning attempts to create models that can then be trained to perform a task without providing them a strict definition for how to approach the problem, like a human learning.[2]

The field of Machine Learning seeks to develop such models that can be trained on data using a training algorithm. The model during training is expected to detect patterns that it can use to take more effective actions or make more accurate predictions. The patterns that the model can detect after training are not always clear to the developer of the model, this has resulted in the term of the Black-Box where we input data and get results without knowledge of the boxes inner workings. Over the years multiple models and training algorithms have been developed, each with their own strengths and weaknesses.

In a lot of applications Machine Learning models show promising results when tasked with predicting possible courses of action, responses to certain situations, and future trends. Furthermore, Machine Learning models have very high accuracy in the field of image and voice recognition, so much so that they have become the primary approach to solving such problems. This is why researchers are now trying to find if these approaches can benefit the field of threat detection in computer networks.

3.1.3 Neural Networks

Neural Networks are a type of model used in the field of Machine Learning. They attempt to simulate the structure of a biological brain using concepts such as Neurons and Synapses. [3]

A Neural Network is a collection of Neurons that are connected to each other, depending on the topology of the model the connections can be different, these connections are also weighted which means that the connection has a set amount of influence to the next Neuron. The Neural Network structure is inspired by the biological structure of the brain.

To train these models special algorithms are used along with vast amounts of data. During training the weights of connections are altered to either strengthen or weaken the influence of the connection. After training, the accuracy of the model is calculated using test data which the model has not seen before. This methodology attempts to emulate the way a biological brain functions when learning, like a student solving mathematical problems with known solutions and then can proceed to solve problems with unknown solutions.

Neural Networks are also flexible in terms of structure and multiple topologies and training algorithms have been defined over time to serve specific purposes. Some Neural Networks are more effective with homogenous data such as images and sound, while other Neural Networks are better with Time-Series data such as the temperature reading of a sensor.

3.1.4 Computer Networks

A Computer Network is a collection of computers that can communicate with each other to share resources. The computers that make up the network are usually referred to as nodes.

Computer Networks can employee different schemes to connect the nodes together, the methodology used to define the structure is referred to as the Network topology. Connections can be made using either electrical wiring or optical wiring, wireless connections are also possible using Electromagnetic Waves and Radio-Frequency. Furthermore, to facilitate communication between nodes a protocol needs to be defined and each node needs to use this protocol to understand each other. Nodes in the network can also be grouped into categories depending on their primary purpose, common categories include Servers, Clients, and Hosts. A Servers primary purpose is to serve data to Clients, Clients themselves can be other Servers or Nodes. A Host refers to a simple Node in network but can also refer to a Node who is running or *hosting* a service that other Nodes can make use of.

The field of Computer Networks is vast and from it we have managed to create networks which together connect the whole world into a single large Network of Networks known as the Internet.

3.1.4.1 Network Traffic

Network traffic represents the messages that are communicated between nodes in a Computer Network as they travel between Nodes. Messages are often split into smaller pieces during communication, we refer to these pieces as Packets. Depending on the Network Protocol, the structure of the packets can vary to suit the needs of the protocol. The procedure of capturing Network traffic is called Packet Sniffing or Packet Snooping, this is usually done using specialized software.

3.2 Required Technologies

3.2.1 Wireshark

Wireshark is a free and open-source packet sniffing software. It also provides tools for analysing the captured packets. It is widely used to troubleshoot computer networks and to assist in the development of communication protocols. It supports most network protocols and can also decrypt data when the network protocol makes use of encryption algorithms. Furthermore, it provides the ability to filter captured data and can also save the data in multiple formats.

Wireshark can be used via a Graphical User Interface or a command-line environment. It is the preferred choice for packet sniffing and has been in development since 1998 using the C programming language [12].

3.2.2 MySQL

MySQL is an open-source relational database management system owned by the Oracle Corporation. It has been in development since 1995 and is written in the C and C++ programming languages.

A relational database is a type of database in which data is stored in tables which represent entities. Tables can be related to each other using foreign keys, these relations represent relationships between entities such as an Author and his books. Data can be extracted, modified, and inserted to and from the database using a unique language called SQL which stands for Structured Query Language. Operations in SQL are done using statements and a collection of SQL statements is called a query.

Due to its open-source nature and reliability MySQL has been used widely as a long-term data storage solution. MySQL is also a main component in the LAMP web application software stack alongside the Linux Operating System, Apache Web Server, and the PHP programming language. [19]

3.2.3 PowerShell

PowerShell is an open-source tool developed and maintained by the Microsoft Corporation. It is used for task automation using a command-line shell, a scripting language, and a configuration management framework. PowerShell was originally available only on Windows, but it is now available for Linux and MacOS. PowerShell was developed using the C# programming language using the .NET Core framework.

Developers can use PowerShell to automate tasks, build, test, and deploy solutions. PowerShell can also serve as an alternative command-line shell. The scripting language has support for functions, classes, modules, and native support for common file formats such as CSV, JSON, and XML.

PowerShell modules are packages that contain other PowerShell components, these modules can be shared with other users and developers using online repositories. [20]

3.2.4 JSON

JSON, which stands for JavaScript Object Notation, is a file and data format. It is an open standard and it is written as text. It was originally derived from the JavaScript programming language but is now used in other programming languages for multiple tasks.

Data in JSON format is a collection of objects containing key-value pairs, and arrays. JSON is the primary data format used in web applications to exchange data. It can also be used as a format to store configurations. JSON files use the “.json” extension. [13]

3.2.5 XML

XML stands for eXtensible Markup Language is a language and file format used to store, transmit and serialize data. It is an open standard specified by the World Wide Web Consortium. The standard defines a ruleset for document encoding in a text format that can be readable for both humans and machines. XML uses tags to define data structures, tags come in pairs with a starting and closing tag, between the start and end tag is the data. XML tags can be nested and can also contain metadata and attributes.

It has been used as a data exchange format on the web and to define documents. XML is another data format often used to store configurations for applications. [15]

3.2.6 CSV Files

CSV, which stands for Comma Separated Values, is a file format used to store tabular data. Each line in a CSV file represents a record, values are separated using commas, and the first line in the file is used to define the fields. The file format does not have any published standard.

CSV files are a common way to share tabular data. They are used by users to share data and is also a common file format used to exchange data between applications. It can also be used to import data to spreadsheets and Databases with native support in most applications thanks to its simple format that requires minimal processing to be parsed. [14]

3.2.7 C# Programming Language

C# is an open-source object-oriented programming language developed and maintained by the Microsoft Corporation. It uses type-safety and has a C style syntax like other programming languages in the C family of languages. It has native support for exception handling, nullable-types, lambda expressions and asynchronous operations. C# also has a built-in feature called LINQ which stands for Language Integrated Query. This feature provides the developer with tools with which he can query data sources.

C# programs run on the Common Language Runtime or CLR which is a virtual system that allows the execution of C# programs. When the program is compiled, it is transformed into an Intermediate Language that can then be executed on the CLR with Just In Time compilation to convert the Intermediate Language into machine code. C# is most used in conjunction with the .NET Core framework.[6]

3.2.8 .NET Core

The .NET Core is an open-source development platform developed and maintained by the Microsoft Corporation. It can be used to build Web Applications, Desktop Applications, Windows Applications, and services, IoT applications, and Machine Learning applications.

It has an extensive library of functionalities that can help in the development process. It has built-in support for a garbage collector, Attributes, asynchronous code, Events, Generic types, Parallel programming, and Exceptions among others. Applications developed using the .NET Core framework are also cross-platform, meaning that they can run in multiple Operating Systems such as Android, Windows, and Linux. Furthermore, it can run on x86, x64, and Arm64 architectures. There is also support for emulated environments.

Applications built using the .NET framework can be written in the C#, F#, and Visual Basic programming languages. All these programming languages are developed and maintained by the Microsoft Corporation. [7]

3.2.9 Java

Java is an object-oriented programming language developed and maintained by Sun Microsystems, a subsidiary of the Oracle Corporation. It launched in 1995 and has seen widespread adoption.

Java programs are compiled to bytecode which can then be run on the Java Virtual Machine using Just-In-Time compilation. This feature also allows the programmer to “*write once, run everywhere”* using the same bytecode. Memory management is handled by the garbage collector and the syntax is similar to other languages in the C family of programming languages. [8]

3.2.10 CiCFlowMeter

CiCFlowMeter is a program written in the Java programming language that can generate flows from network traffic. It has been developed by researchers at the Canadian Institute for Cybersecurity and is open-source software.

It can be used to extract features from network traffic and can also export the results to CSV files. It can be run using a Graphical User Interface or a command-line environment. Furthermore, it allows the user to directly read and process traffic from the network or use already captured network traffic stored in .pcap files.[9]

3.2.11 Erlang

Erlang is a programming language that was originally developed in the Ericsson Computer Science Laboratory. It can be used to create large scale real-time systems that require high availability. It has seen extensive use in the telecom, banking, e-commerce, and telephony industries. It launched officially in 1986 and has become an open-source technology since 1998.

It is a functional language and is focused on concurrency. Erlang is often mentioned alongside the Open Telecom Platform or OTP, a platform made up of a runtime system for Erlang programs and a library of components written with Erlang. [10]

3.2.11 RabbitMQ

RabbitMQ is an open-source message broker software from VMware Inc. It has been developed using the OTP framework and is written in the Erlang programming language. It has native support for multiple messaging protocols such as Advanced Message Queuing Protocol-AMQP for short, Streaming Text Oriented Messaging Protocol-STOMP, and MQ Telemetry Transport among others.

It has seen widespread adoption in the industry, from small to large-scale solutions. It supports multiple operating systems and can be deployed in the cloud or on premises. RabbitMQ can also be deployed in either a distributed or federated configuration for high availability and scalability.

Using RabbitMQ a developer can define queues to which messages can be sent and then received by client applications. Libraries exist in most programming languages that contain the necessary functions to use RabbitMQ. [11]

Chapter 4

Requirements and Specifications

4.1 Modular Design

Before proceeding to defining any requirements, the main modules of the project need to be decided. The system can be divided into three main components. The first component is data collection, the second component is data processing, and the final component is threat detection. There is another complementary component, that is data storage. The requirements for each component will be discussed separately.

4.2 Requirements

Requirements are split into two categories. Functional and Non-Functional. Functional requirements represent specific functionalities, features, and behaviors that a system or application should implement to achieve its intended purpose. These requirements define what the system must do and how it should respond to user interactions or other events. They also serve as blueprints during development, providing a clear understanding of the system's core functionalities. They are also reference points during the testing phase of the system to ensure that the system is behaving according to these requirements.

On the other hand, non-functional requirements focus on the qualities and characteristics of a system, rather than its specific functionalities. These requirements represent aspects such as performance, security, usability, reliability, scalability, and maintainability. Non-functional requirements describe how well the system should perform in terms of speed, responsiveness, availability, and other attributes.

4.2.1 Functional Requirements

4.2.1.1 Data Collection Component

The component that is responsible for data collection should capture network traffic from the target network and forward it to the data processing component. The period of the capture, capture interval, network to perform capture, and input of the next component should be configurable by the user. The component should also allow the user to configure filters that can filter data during the capture process. The component should also support data transfer through a network in case the data processing component is running on a remote host.

Capture Network Traffic: The data collection component should be capable of capturing network traffic from a target network. It should be able to collect packets and forward them to the data processing component for further analysis. The component should support configurable capture parameters such as the capture period, capture interval, and the specific network on which network traffic will be captured.

Configurable Data Filtering: The data collection component should provide the ability to configure data filters during the capture process. Users should be able to define filter criteria based on specific network protocols, IP addresses, ports, or other parameters. These filters should allow for the selective capture of network traffic based on the defined criteria, allowing the user to focus on specific subsets of data for analysis.

4.2.1.2 Data Processing Component

The Data processing component should process network traffic and extract features from it. Network traffic will come in the form of packets and the component should present the extracted features in tabular format. The user should be able to configure the list of features to be extracted. Upon successful processing of network traffic, the component should also forward the results to the threat detection component.

Feature Extraction: The component should be able to process network traffic packets and extract features from them. The features to be extracted should be configurable by the user. Examples of features could include source and destination IP addresses, port numbers, packet size, protocol type, etc.

Tabular Presentation: The extracted features should be presented in a tabular format. This could be in the form of a table or a file (e.g., CSV, Excel) containing the extracted feature values. The tabular presentation should be organized and easy to understand.

Configuration of Feature List: The user should have the ability to configure the list of features to be extracted. This could include selecting predefined feature sets or defining custom feature combinations based on their specific requirements.

Integration with Threat Detection Component: Upon successful processing of network traffic and extraction of features, the component should forward the extracted features to the threat detection component. This ensures that the processed data is available for further analysis and anomaly detection.

Error Handling: The component should handle potential errors or exceptions that may occur during the processing of network traffic. Appropriate error messages or logging mechanisms should be implemented to notify the user or system administrators about any issues encountered.

4.2.1.3 Threat Detection Component

The Threat detection component should be able to execute the detection algorithm that has been configured by the user and present its results. The component should provide the necessary infrastructure to integrate and call external detection algorithms, which can include Python scripts, binary executables, or other compatible formats. The key requirement is that the detection algorithm follows the expected input and output specifications defined by the Threat detection component.

Detection Algorithm configuration: The component should allow the user to configure and specify the detection algorithm to be used for analysing the extracted features. This configuration could include providing the path or reference to the algorithm's script or executable, along with any required input parameters or settings. The component should ensure that the algorithm is properly invoked, allowing it to process the extracted features and generate the corresponding threat detection results.

Execution of the Algorithm: The component should be responsible for executing the configured threat detection algorithm. It should handle the invocation of the algorithm, passing the extracted features as input, and managing the execution process. The component should implement error handling to handle any potential issues during the execution, such as invalid algorithm configurations, errors in the algorithm's execution, or unexpected termination. Error messages and appropriate logging should be implemented to facilitate debugging and troubleshooting in case of algorithm execution failures.

Result Presentation: the component should handle the retrieval and presentation of the detection results. This could involve capturing the output generated by the detection algorithm and presenting it in a format that is easily interpretable by the user. The results could be displayed as a summary report, log files, or any other suitable format that provides meaningful insights into the detected threats.

4.2.1.4 Database Component

Data Storage: The database component should provide a reliable and efficient means of storing the collected network traffic data, extracted features, and detection results. It should support the storage of data in a structured manner, preserving the integrity and consistency of the information. The component should allow for the creation and management of appropriate database tables or collections to store different types of data, ensuring optimal organization and retrieval.

Data Retrieval: The component should enable the retrieval of stored data based on various criteria, such as time intervals, specific network traffic attributes, or detection results. It should support query functionalities that allow users to retrieve relevant data for further analysis or reporting. The retrieval process should be efficient, providing timely access to the required data, even with large datasets.

Data Management: The component should offer mechanisms for managing the stored data, including data backup, restoration, and archival procedures. It should provide options for data retention policies, allowing the user to define the duration for which data should be preserved. The component should also support data indexing and optimization techniques to enhance the performance of data retrieval operations.

Integration with Other Components: The database component should seamlessly integrate with the other components of the system, such as the data collection component and the data processing component. It should facilitate the transfer and storage of data between these components, ensuring that the necessary data is appropriately captured, processed, and stored in the database. The component should handle data synchronization and consistency across the system to maintain accurate and up-to-date information.

Security and Access Control: The database component should implement appropriate security measures to protect the stored data from unauthorized access, tampering, or disclosure. It should support user authentication and authorization mechanisms to control access to the database and its data. Additionally, the component should adhere to data privacy regulations and best practices, ensuring the confidentiality and integrity of the stored information.

4.2.2Non-Functional Requirements

4.2.2.1 Data Collection Component

- **Efficiency**: The Data collection module should be designed to operate efficiently with minimal resource consumption, ensuring it runs smoothly on computers with limited resources.

- **Low Overhead**: The module should introduce minimal overhead to the system, and the overall performance of the capturing device.
- **Device Compatibility**: The module should be compatible with a wide range of devices, allowing for seamless integration and data capture from various sources.
- **Protocol Support**: The module should support multiple network protocols, ensuring the ability to capture data from different types of networks.
- **Minimal Logging**: The module should implement logging capabilities for troubleshooting purposes only, minimizing unnecessary log entries to reduce resource usage.
- **Troubleshooting Information:** The logs should provide relevant information related to the capture and forwarding process, assisting in identifying and resolving any issues that may arise.
- **Detailed Documentation**: The module should be accompanied by extensive and detailed documentation, providing clear instructions on configuration options, command-line usage, and advanced capabilities such as filtering and network selection.
- **Configuration Simplification**: To accommodate users with limited resources or command-line access, the module should support configuration through XML or JSON files, providing a streamlined and efficient configuration process without the need for a graphical user interface.
- **Compliance**: The module should adhere to relevant data privacy regulations and best practices to ensure the privacy of captured network data.

4.2.2.2 Data Processing Component

- **Processing capacity**: The component should be able to handle varying volumes of network traffic data efficiently, scaling according to the size of the network(s) it is processing.
- **Real-time processing**: The component should have minimal processing time to ensure timely analysis of the network traffic data.
- **Scalable architecture**: The component should be designed with scalability in mind to handle increasing data volumes and potential expansion of the network(s) it processes.

- **Feature extraction customization**: The component should allow users to select and customize pre-defined features to be extracted from the network traffic data.
- **Configuration using XML or JSON**: The component's configuration should be easily customizable and manageable through XML or JSON files.
- **Input data format**: The component should support input data in the form of packets, preferably in a file format like .pcap.
- **Output data format**: The extracted features should be presented in a tabular format, preferably in a file format like .csv.
- **Error handling**: The component should implement robust error handling mechanisms to handle exceptions.
- **Minimal logging**: Logging should be kept to a minimum, primarily for troubleshooting purposes, to avoid unnecessary resource usage.
- **Graceful failure**: In the event of a failure or error, the component should exit gracefully and provide clear notifications to the user through the log.

4.2.2.3 Threat Detection Component

- **Configuration using XML or JSON**: The component's configuration should be easily customizable and manageable using XML or JSON files.
- **Algorithm flexibility**: The component should be flexible in its algorithm invocation methods, allowing users to make use of any technology or programming language they need to develop their detection algorithm.
- **Input data format**: The component should support input data in the form of a CSV file, providing a dataset of extracted features from network traffic.
- **Output data format**: The component should support generating output in a structured format, such as CSV files or JSON files, depending on the algorithm's requirements.
- **Extensibility**: The component should support the addition of new detection algorithms, allowing the system to adapt and incorporate emerging technologies and methodologies.
- **Error handling**: The component should implement error handling to handle exceptions and unexpected situations.

- **Logging**: Logging should be implemented to allow troubleshooting and analysis of the detection algorithm execution.
- **Detailed documentation**: The component should provide extensive documentation, guiding users on configuration, and understanding the input/output formats.

4.2.2.4 Database Component

- **Data Persistence**: The database should ensure the durability and persistence of stored data, even in the event of a system failure or power outage.
- **Reliability**: The database should be reliable, providing consistent access to the data and minimizing any risk for data loss or data corruption.
- **Availability**: The database should support high availability, allowing continuous access to the data and minimal downtime.
- **Data Security**: The database should provide mechanisms to secure the data, protecting against unauthorized access.

4.3 Specifications

Within this section, the specifications of the system will be outlined, providing an understanding of its requirements, functionalities, and limitations. General specifications that apply to the system as a whole will be discussed first, including the expected users, use case diagrams, limitations, and development principles. This will provide context for the system and its intended purpose.

Following the general specifications, more specific specifications for each component of the system will be discussed. By addressing each component individually, a detailed analysis of their functionalities, interfaces, and requirements can be provided. This will enable a comprehensive understanding of the system's architecture and the role of each module within it.

Through these specifications, a clear and well-defined outline of the system will be provided, ensuring that all stakeholders have a solid understanding of its capabilities, limitations, and intended usage.

4.3.1 User Characteristics

The users of the system are mentioned in detail in this section:

- **Algorithm Researchers**: These users are interested in developing and testing new detection algorithms for network threat detection. They would utilize the system's flexibility and customization capabilities to experiment with different approaches and evaluate the performance of their algorithms.
- **Network/System Administrators**: These users are responsible for deploying and the system. They would configure and deploy the system, ensuring its integration with other components.

4.3.2 Development Principles

By incorporating these development principles into the project, it will be better equipped to address the unique requirements and challenges of such a system.

1. **Modularity**: A modular design allows for the independent development and testing of each component, enabling easier maintenance and code reuse. It allows researchers and developers to focus on specific functionalities without affecting the entire system.

2. **Flexibility**: The system needs to be flexible and allow for different detection algorithms and configurations. Researchers and algorithm developers may have different requirements and preferences in terms of the technologies and tools they want to use. Providing flexibility in the system's design allows them to leverage their preferred programming languages, frameworks, or even pre-existing models, promoting innovation and experimentation.

3. **Extensibility**: The system should be designed with extensibility in mind to support future improvements and integrations. As new network protocols emerge or machine learning techniques evolve, the system should be able to adapt and incorporate these advancements without requiring extensive re-architecting. This ensures that the system remains up-to-date and relevant over time.

4. **Scalability**: Scalability is crucial as the system needs to handle varying volumes of network traffic efficiently. By optimizing resource utilization, and leveraging parallel processing techniques, the system can handle increasing amounts of data while maintaining performance and responsiveness.

5. **Usability**: Well-documented configuration files make it easier for researchers and network administrators to set up and operate the system. Clear instructions, informative logs, and error handling mechanisms allow users to understand and troubleshoot the system effectively.

6. **Performance**: Performance is crucial for real-time or near-real-time processing of network traffic. Optimized algorithms, data structures, and efficient processing techniques should be used to ensure that the system can keep up with the data flow and deliver threat detection results with minimal latency.

7. **Reliability and Fault Tolerance**: The system should be resilient to failures and errors to ensure seamless operation. It should handle unexpected scenarios gracefully and recover from failures to minimize disruption. Reliability and fault tolerance mechanisms are important to maintain the system's availability and ensure a steady flow of data processing.

8. **Security**: While data privacy may not be a primary concern for this project, implementing secure coding practices and appropriate security measures is still important. This ensures that the system is protected against potential vulnerabilities and safeguards it from unauthorized access or malicious activities.

9. **Documentation**: Extensive and comprehensive documentation is necessary for researchers and developers to understand the system's functionality, configuration options, and usage guidelines. Well-documented code and technical guides allow for easier troubleshooting,

and future development. Clear documentation helps users leverage the system's capabilities effectively.

4.3.3 Design Restrictions and Limitations

The architecture and design of the system has the following limitations and restrictions:

1. **Centralized Data Collection**: The system requires a centralized collection point within the network architecture to capture all network traffic. This requires the identification of a suitable location, such as a central switch or network tap, to avoid partial data collection. It should be noted that this requirement may limit the deployment of the system to networks that allow for a centralized collection approach.
2. **Access to Network Infrastructure**: To capture network traffic, the system requires access to the network infrastructure, including the necessary permissions and credentials. This may require collaboration and coordination with network administrators or relevant stakeholders to ensure the system's deployment and operation within the network.

3. **Availability of Compatible Network Protocols**: The system relies on the availability and support of compatible network protocols for capturing and processing network traffic. It should be compatible with common network protocols used in the target network environment, such as TCP/IP, Ethernet, or Wi-Fi protocols.

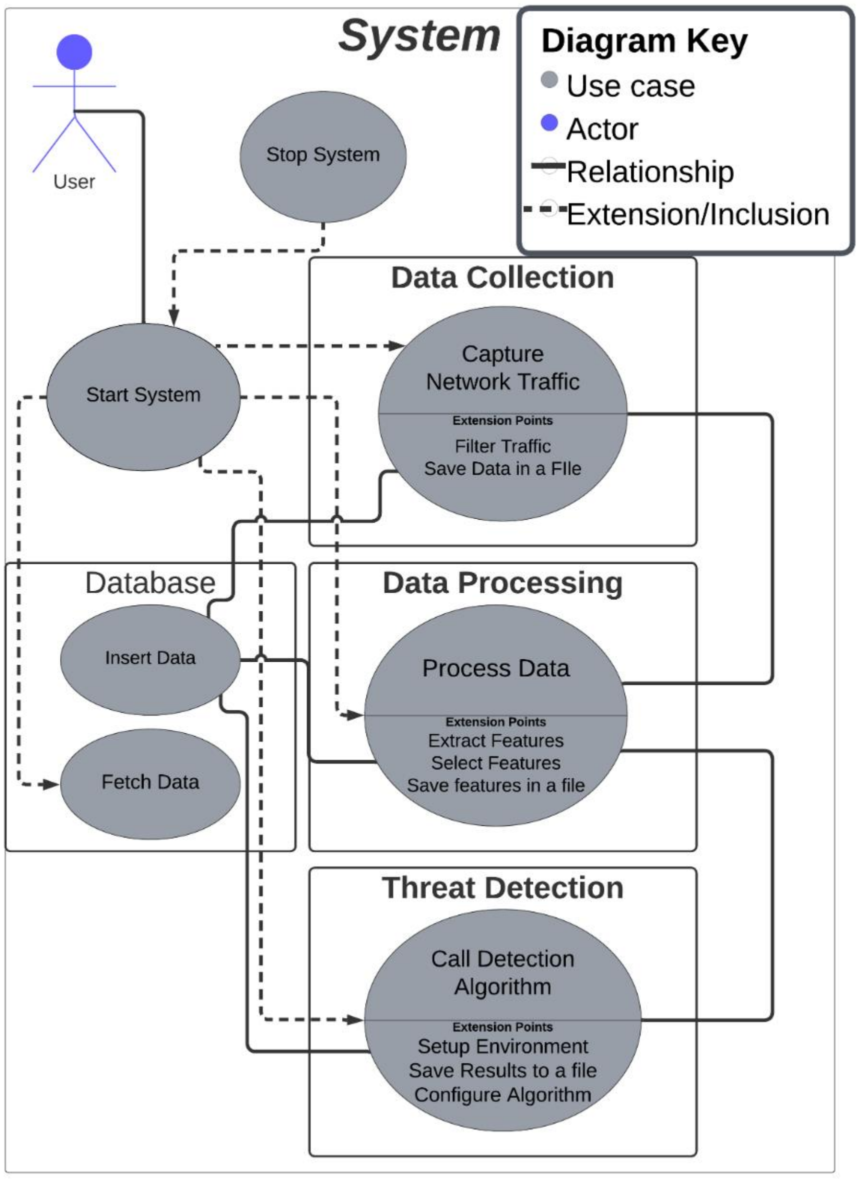
4. **System Compatibility**: The system's components may have compatibility requirements with the operating systems, programming languages, and dependencies used for implementation. It is essential to consider the compatibility constraints of each component and ensure the necessary environment is available for the system's deployment.

5. **Hardware and Resource Requirements**: The system's performance and functionality may be affected by hardware and resource limitations. Considerations should be given to the hardware specifications, processing

power, memory, and storage requirements to ensure the system can effectively handle the anticipated workload and data volumes. It is also important to consider that the hardware requirements can vary depending on the scale of the network and the amount of traffic it produces.

6. **Data Privacy and Legal Compliance**: The system should abide to all data privacy regulations and legal requirements when capturing and processing network traffic data.

4.3.3 Use Case Diagram



Chapter5

System Design and Implementation

5.1 System Architecture

The system follows a distributed architecture, consisting of multiple interconnected components that work together to capture, process, and detect threats in network traffic. The main components include the Data Collection component, Data Processing component, Threat Detection component, and the Database component.

The Data Collection component is responsible for capturing network traffic. It utilizes appropriate mechanisms to intercept and capture the traffic. Once the data is captured, it is forwarded to the Data Processing component. The Data Processing component exposes an API endpoint to receive the data.

The Data Processing component receives the captured data from the Data Collection component through an exposed API endpoint. The API receives data using HTTP POST requests. When the API receives data, it is inserted to the Database component and a message is sent to the appropriate queue with the newly inserted record id. When the Data Processing component receives a message in the queue it retrieves the data record from the database component. Then it performs necessary preprocessing and feature extraction. The processed data is then sent to the Threat Detection component for analysis. Communication between the Data Processing and Threat Detection components is also facilitated through RabbitMQ message queues. The Data Processing component inserts the processed data in the Database component, and a message with the newly inserted record id is sent to the appropriate queue.

The Threat Detection component is responsible for invoking the configured detection algorithm on the processed data. It utilizes the extracted features to perform threat analysis and generate detection results. The detection algorithm can be customized by the user. The Threat Detection component awaits messages from a queue. When the queue receives a message, the component will then retrieve the Processed Data record with the id provided from the message. Then the Detection Algorithm is invoked, and the results are inserted to the Database component, finally a message with the newly inserted record id is sent to the appropriate queue.

The Database component is an integral part of the system, providing long-term storage for the captured data, processed data, and detection results. The use of the Database component ensures data persistence and enables future analysis and retrieval.

Overall, the system architecture enables efficient data collection, processing, and threat detection by utilizing RabbitMQ message queues for communication between the components. The integration of the Database component ensures long-term storage and retrieval of the captured data and detection results, providing a comprehensive solution for network traffic analysis and threat detection.

5.2 Data Capture Component

The Data Capture module must perform three tasks. First, it must read the configuration provided by the user, then it must capture traffic from the network according to the configuration, and finally it must forward the traffic that was captured to the Data Processing component.

5.2.1 Component Design

Considering that the Data Capture component is likely to be deployed on a computer with limited processing and memory resources, a script-based approach is used. Using the script, we can make use of external tools to capture traffic. Then the script can make an HTTP Request to a remote API to forward the data for storage and processing. The script can also be automated using tools like Cron Job and Windows Task Scheduler. With this approach, the component is kept simple and uses as little resources possible.

5.2.2 Component Implementation

The Data capturing component is implemented using PowerShell scripting language. PowerShell provides the necessary functionalities to interact with the operating system, execute commands, and handle the capture process. The component uses PowerShell functions to read the configuration file, execute the tshark command, and make the HTTP POST request.

The Data capturing component integrates with Wireshark, specifically utilizing the tshark command-line tool. By invoking tshark, the component can capture network traffic and save the captured traffic to a pcap file before sending it to the next component.

Configuration is provided using an XML file. Thanks to a built-in functionality of PowerShell no additional modules are needed to read XML files.

5.2.3 Requirements

Thanks to the minimal design, the Data Capture component has only two requirements:

1. PowerShell Scripting Language v7.0 and later
2. Wireshark

5.2.4 Deployment and Usage

The process of deploying the Data Capture component is as follows:

1. Copy the directory of the component to the desired location.
2. Apply the desired configuration using the provided XML file. Details about the configuration options are provided in the README file.
3. (Optional) Automate the execution of the capture script using tools like Cron Job and Task Scheduler. Blueprints for these tasks are provided as separate scripts.

The component can be used by executing the capture script provided. The capture script is a PowerShell script.

5.3 Data Processing Component

5.3.1 Design

The Data Processing component is the largest of the three main components. It must perform multiple tasks. The primary task is the processing of Data as it is received from the Data Capture component.

Considering that the Data Capture component is likely to be running on a remote computer, an API must be implemented and exposed to the network to facilitate the transmission of data. The API must receive the data, insert it into the Database Component and send a message to begin processing the data. The message must contain the record ID of the data inserted into the Database Component.

For a more scalable design a message brokering service has been integrated into the system. This allows for multiple processing events to happen simultaneously and with the addition of failure queues there is an additional layer of fault tolerance. Using the queue system of the message broker service we can define queues where messages can be sent and received from our components. These queues can act as a buffer between components, this way each component can be independent from each other. Furthermore, each component can be scaled according to its processing needs, for example, if the processing component is overwhelmed by incoming data, the queue where the captured data messages are sent will grow, allowing the Data Capture component to continue its operation, the Data Processing component can then be scaled to handle the increased flow of incoming data.

To utilize the Queue system of the Message Broker service a Consumer Model is used. A consumer is an entity that subscribes to a queue. When a message is sent to that queue the Message Broker service will signal a consumer that is subscribed to the queue. If multiple consumers are subscribed to the same queue, then the consumer will be selected using a round-robin approach. When a consumer is signaled by the message broker it will execute a method which can be defined by the developer. This method has access to the message that has been sent.to the queue.

The Data Processing is performed using one or more consumers. The consumer is subscribed to the queue where the raw data is sent. When a message is received the consumer reads the message to get the ID of the record inside the Database component, then the raw data is fetched from the Database component. After fetching the raw data, a tool for feature extraction is used to process the data, when the tool is finished processing the raw data the results are saved into a temporary location before being inserted into the Database component. During the insertion of processed data, the ID of the related raw data must be provided to establish the relationship between the raw and processed data inside the Database component. Finally, when the processed data has been inserted into the Database component a message is sent to the appropriate queue. The message contains the ID of the newly inserted record of processed data.

The number of consumers can be configured by the user. Depending on the amount of raw data produced by the Data Capture component, the user can increase or decrease the number of consumers that are used, offering flexibility and scalability.

5.3.2 Implementation

The Data processing component has been developed using the .NET Core framework and the C# programming language. The Messaging Brokering Service that is sued is RabbitMQ. The component is designed as a console application without a Graphics User Interface.

The API that receives data from the Data Capture component has also been developed using the .NET Core framework and the C# programming language. The API is designed as a Web Application.

Access to the Database component is offered through a shared library of C# classes that both the API and Data Processing application use. It contains classes and methods that can provide access to the Database Component along with other functionality that is shared between the two applications. This was done for code-reuse purposes. The classes that give access to the Database Component have all the necessary methods that allow the applications to connect to the Database, query the Database, and insert new data to the Database.

The raw data consumers are extensions of the generic consumer class provided by the RabbitMQ library. Additionally, helper methods have been defined to assist consumers with the exchange of messages and the declaration of queues. Queues can be configured by the user, allowing for a longer or shorter message lifespan known as Time-To-Live or TTL. When the consumer receives a message, the raw data is fetched from the Database component using the record ID inside the message, then using an external tool named CiCFlowMeter, the features are extracted from the raw data and saved inside a CSV file. When the extraction of the features is finished, the consumer will insert the processed data in the Database Component, a relationship with the processed and raw data is created using the record ID of the raw data to reference the origin of the results. Finally, when the processed data has been inserted successfully into the Database component, a message is sent to the appropriate queue signalling the next component to begin.

Exception handling has also been implemented for a more fault-tolerant application. Furthermore, in the case of a failure, the application will exit gracefully and provide the user with a detailed log of the incident that caused the failure to assist in debugging. Finally, the connection parameters for the Message broker service and the Database Connector are configurable by the user with the use of JSON files.

5.3.3 Requirements

For the deployment of the Processing component the following requirements must be met.

1. Valid and running Installation of RabbitMQa.
2. Valid and running installation of a MySQL server. The MySQL server will contain the Database instance of our system and represents the Database component. Further details will be provided in a later section.
3. The user must allow communication between the computer/server pairs that host the components. Note that if needed, all components can be hosted on the same computer/server.
    a. Data Capture - Data Processing web API
    b. Data Processing web API - Data Processing application
    c. Data Processing web API – Database
    d. Data Processing application – Database
4. (Optional) .NET Core Runtime version 7.0

5.3.4 Deployment and Usage

Deployment of the Data Processing is done by following the procedure below.

1. Build the application using .NET Core or use a pre-built package.
    a. Optionally, the user can build the application and use the framework dependent executable, the executable will be in the form of a DLL file. However, this requires the user to have the .NET Core Runtime v7.0 installed.
2. Copy the directory of the packaged application to the desired location.
3. Using the provided JSON files, apply the configuration needed. The configuration includes the connection parameters for the Message Brokering Service-RabbitMQ, the Database Component, the path of the CiCFlowMeter executable, and the output path for the processed data. Detailed explanation of the configuration options is provided in the README file.

4. After applying the configuration, launch the executable file. Any warnings or errors will be displayed inside the console window of the application. If the user has chosen to use the framework dependent executable, the application can be launched using the dotnet run command and providing the path to the DLL file.

Deployment of the API module is done separately. This is done for modularity. Another reason is that the API can be deployed with the Database component on the same server for lower latency. The procedure for deployment is like the Data processing module.

1. Build the Web app using .NET Core or use a pre-built package for the target system.
    a. Optionally, the user can build the Web app and use the framework dependent executable, the executable will be in the form of a DLL file. However, this requires the user to have the .NET Core Runtime v7.0 installed.
2. Copy the directory of the packaged web app to the desired location.
3. Using the provided JSON files, apply the configuration needed. The configuration includes the connection parameters for the Message Brokering Service-RabbitMQ, the Database Component, and the names of the queues where messages for successful or failed transmissions are sent. Detailed explanation of the configuration options is provided in the README file.
4. After applying the configuration, launch the executable file. Any warnings or errors will be displayed inside the console window of the application. If the user has chosen to use the framework dependent executable, the application can be launched using the dotnet run command and providing the path to the DLL file.
5. The console window will provide the necessary information to access the API, such as URL and port number. The user can configure these variables inside the configuration file; however, this is optional.

5.4 Threat Detection Component

The Threat Detection Component is responsible for the execution of the detection algorithm that is provided by the user. It must make sure that the execution process goes smoothly, and it is also tasked with inserting the results of the algorithm to the Database Component. The component must provide the user with the tools needed for the deployment of the detection algorithm of choice. The component limits the user only in the format of the expected input. The algorithm must take as input a CSV file with features extracted from network traffic.

5.4.1 Design

The Threat Detection component follows a design pattern similar to the Data Processing component. For this reason, the component has been merged with the Data Processing component in a single application.

Like the Data Processing component, the Threat Detection component defines a custom consumer class which is subscribed to a queue. When the queue receives a message, a processed data record is fetched from the Database component using the record ID inside the message. When the record is fetched it is stored as a temporary file in CSV format, then the Threat Detection algorithm that was defined by the user inside the configuration will be executed with the temporary file containing the processed data as input. When the algorithm has been executed successfully, the results are inserted into the Database component, and the processed record ID is used to create a relationship between the data in the Database.

The number of consumers can be configured by the user. Additionally, the user must provide the path to the algorithm executable along with the command used to execute the algorithm and any other arguments that are needed.

5.4.2 Implementation

The component has been developed using the .NET Core framework and the C# programming language. It is packaged alongside the Data Processing component due to their identical design and requirements. The consumer class is an extension of the generic Consumer class provided by the RabbitMQ library. Access to the Database component is provided by the Shared Library used by the previously discussed components. The component also contains a tool that allows the user to test if the provided configuration is valid and that the Threat Detection algorithm can be executed successfully, this tool is provided to assist the user in debugging the Detection Algorithm before deployment.

Error handling has been implemented for increased fault-tolerance. Furthermore, all errors that are encountered are logged to assist the user in debugging.

5.4.3 Requirements

For the deployment of the Threat Detection component the following requirements must be met.

1. All Requirements of The Data Processing Component
2. Any requirements that must be met to execute the Threat Detection algorithm that the user has provided. This can be the Runtime environment of the programming language of choice, like the case of a language like Java or C#.

5.4.4 Deployment and Usage

The component is deployed alongside the Data Processing component and as such no additional steps need to be taken, other than the configuration of the application. A detailed description of the configuration options is provided in the README file of the application.

The component is launched alongside the Data Processing component, no additional executables are needed. The console window will provide information regarding the startup process of each component, and all errors will be logged inside a log file.

5.5 Database Component

The Database component is responsible for the long-term storage of data in all its forms. It is also responsible for providing access to the data when needed using queries. In addition, the Database component is tasked with the creation of relationships between data, these relationships represent the transitions between each form of data, this includes raw data, processed data, and results from the detection algorithm. These relationships allow the user to trace each result back to its original source. This can potentially help in the analysis of the results.

5.5.1 Design

The Database component has two primary functions, the storage of data long-term, and the querying of stored data. In addition, a mechanism to establish relationships between data records is needed. The database must be able to store binary data such as captured network traffic. Finally, the Database component must perform its primary functions in the shortest amount of time possible, this is critical for scalability, and it is even more crucial when considering the real-time nature of the system.

Each form of data, that is raw data, processed data, and results from processed data, must be a different type of record. Additionally, each record must be uniquely identifiable by an ID, this unique is called the Primary Key. The unique ID of each record will be used to reference other records for the establishment of relationships between data, these references are called Foreign Keys.

The Database component must also be accessible through the network to be accessible to components that are running on remote hosts. Finally, the component must provide an interface that other components can use to perform data insertions and data querying.

5.5.2 Implementation

The Database component has been implemented using as Relational Database using the MySQL Relational Database Management System. Records in the database are represented as lines inside a table, each record can have multiple values, where each of these values is associated with a field. Fields are represented as columns of a table. A table represents an entity, the case of our system our entities are the data in all its forms. There are three tables in total, the raw data table, the processed data table, and the results table.

Each record in the raw data table has the following values, record ID, insert date, and data. The record ID is a numerical value that must be unique for each record, and it is used to identify a record. The insert date is a date time value that notes the exact date and time the record was inserted into the database. The data value is binary data, this binary data represents the contents of the captured network traffic data file.

The processed data records have the same values as the raw data records, the only addition is a value that references the record ID of the raw data that was used to create the processed data record, in other words the processed data record has a foreign key for a raw data record.

The results records are structured like the processed data records with the only difference being that they reference processed data records and not raw data records. Using this scheme each result has a processed data record where it originated from, and that processed data record also has a raw data record from which it originates, thus establishing a traceable origin path for each result.

Binary data inside the database is stored as a Blob type. The blob type does not contain any metadata such as file name or file extension. This means that the developer and user must be aware of the nature of the stored data, such as file format and file extension.

5.5.3 Database ERD

The following Entity-Relationship-Diagram is a visual representation of the entities inside the database and their relationships with each other.

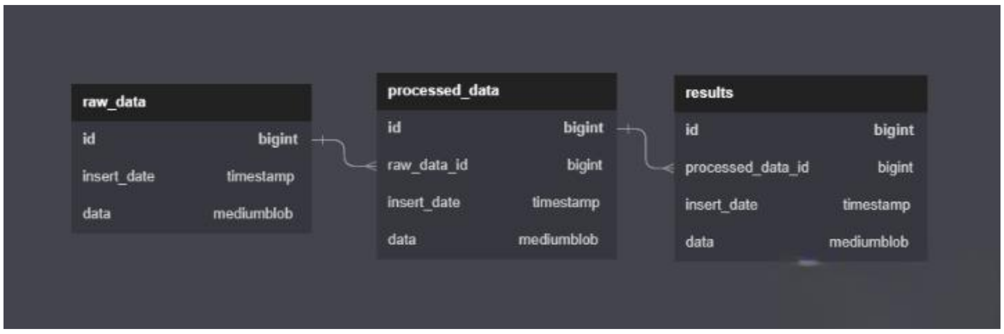


Figure 4 Database ERD

5.5.4 Requirements

There are no special requirements for the deployment of the Database component, however, it should be noted that for optimal performance the component should be hosted on a computer or server that is not severely limited in terms of processing and memory resources. In addition, the Database component must be available to the network to be accessible to other components and the link must be able to handle the amount of traffic that will be generated. If any of the resources is not adequate, there can be negative effects on the performance of the component and by extension the system.

5.5.5 Deployment and Usage

Deployment of the component is done with the following procedure.

1. Install MySQL Server on the host machine.
2. Connect to the Database server using the default user credentials. This can be done from a command-line environment or through a Graphical User Interface by utilizing an external application named MySQL Workbench.
3. Create a new Database Instance. The name of the Database instance can be defined by the user. This instance name is part of the configuration that must be applied to the other components that need to have access to the Database component.
4. Using the provided SQL query, create the tables of the database. The provided SQL Query contains the necessary statements to create all the necessary tables.
5. (Optional but highly recommended) Change the default user credentials and create another user that has access to the database instance with SELECT and INSERT privileges. This user will be used from the components to access the Database Component, the credentials for the user must be provided in the configuration of the components.
6. The user can check if the components can access the database by launching the other components. Each component will attempt to establish a connection to the Database component. If the connection fails, an error will be displayed detailing where the connection attempt was stopped. Using this information, the user can know if the issue is due to a lack of communication between the components, invalid credentials, or any other issue.

5.6 Shared Library

The Shared Library is a collection of Classes and methods that are used in multiple components. These classes and methods are used to establish a connection with the Database component and Message Brokering Service. In addition, helper methods are provided for easier message transmission and data querying. Finally, a class for each entity in the database has been created for easier manipulation of data records.

This Shared Library is linked to each component that makes use of it during the build process and as such no further action is needed from the user.

5.6.1 Design and Implementation

The Classes and methods that are used for accessing the Database component make use of the Entity Framework. This is a library that allows the developer to map Database entities to .NET Objects. This can be done by using predefined tags inside the class definition, these tags are used to associate the Object with an entity/table and the object properties with columns. The use of the Entity Framework then allows for easier access to the record of the database. Using a specific syntax, the developer can declare methods that perform queries without writing any SQL code. Furthermore, by mapping the records to a .NET Object, their manipulation becomes much easier and decreases the total amount of code needed.

Helper Classes and methods for the Message Broker Service-RabbitMQ are also defined in the Shared Library. These Classes and methods allow for an easy establishment of connection to the RabbitMQ server, and quick declaration of all the required queues using a .NET Configuration object which we can create easily using the JSON configuration file of the application. Methods for sending messages to queues are also available.

Overall, the Classes and methods that are provided by the Shared Library promote code-reuse and reduce the amount of code needed inside the application, which in turn provides a more readable codebase for the developer.

Chapter 6

Conclusion

6.1 Results

6.2 Summary

6.3 Future Work

REFERENCES

[1]"What is Artificial Intelligence (AI) ? | IBM." https://www.ibm.com/topics/artificial-intelligence

[2] "What is Machine Learning? | IBM." https://www.ibm.com/topics/machine-learning

[3] "What are Neural Networks? | IBM." https://www.ibm.com/topics/neural-networks

[4] "What is an Intrusion Detection System (IDS)? Definition & Types | Fortinet," Fortinet. https://www.fortinet.com/resources/cyberglossary/intrusion-detection-system

[5] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," Cybersecurity, vol. 2, no. 1, Jul. 2019, doi: 10.1186/s42400-019-0038-7.

[6] BillWagner, "A tour of C# - Overview," Microsoft Learn, May 04, 2023. https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

[7] Gewarren, ."NET (and .NET Core) - introduction and overview," Microsoft Learn, Mar. 15, 2023. https://learn.microsoft.com/en-us/dotnet/core/introduction

[8] "Java Software," Oracle. https://www.oracle.com/java/

[9] "Applications | Research | Canadian Institute for Cybersecurity | UNB." https://www.unb.ca/cic/research/applications.html

[10] "About - Erlang/OTP," Erlang.org. https://www.erlang.org/about

[11] "Messaging that just works — RabbitMQ." https://www.rabbitmq.com/

[12] "Wireshark · Go Deep," Wireshark. https://www.wireshark.org/

[13] "JSON." https://www.json.org/json-en.html

[14] "DataHub - a complete solution for Open Data Platforms, Data Catalogs, Data Lakes and Data Management." https://datahub.io/docs/data-packages/csv

[15] "Extensible Markup Language (XML)." https://www.w3.org/XML/

[16] "Snort - Network Intrusion Detection & Prevention System." https://www.snort.org/

[17] "The Zeek Network Security Monitor," Zeek. https://zeek.org/

[18] "Home - Suricata," Suricata, Aug. 10, 2022. https://suricata.io/

[19] "MySQL." https://www.mysql.com/

[20] Sdwheeler, “What is PowerShell? - PowerShell,” Microsoft Learn, Oct. 21, 2022. https://learn.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7.3

APPENDIX A