BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

# ANOMALY RECOGNITION OF EXTENDED DETECTION SYSTEMS

ROZPOZNÁNÍ ANOMÁLIÍ ROZŠÍŘENÝCH DETEKČNÍCH SYSTÉMŮ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR VASIL POPOSKI
AUTOR PRÁCE

SUPERVISOR Mgr. Ing. PAVEL OČENÁŠEK, Ph.D.
VEDOUCÍ PRÁCE

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

# Zadání bakalářské práce

148445

| | |
|---|---|
| Ústav: | Ústav informačních systémů (UIFS) |
| Student: | **Poposki Vasil** |
| Program: | Informační technologie |
| Název: | **Rozpoznání anomálií rozšířených detekčních systémů** |
| Kategorie: | Počítačové sítě |
| Akademický rok: | 2022/23 |

Zadání:

1. Seznamte se s principy analýzy anomálií v prostředí systémů rozšířené detekce (eXtended Detection and Response), respektive systémů detekce průniku jak na síti, tak na koncových bodech.
2. Analyzujte požadavky na systém umožňující analýzu dat z systémů NIPS a HIPS s otevřenou licencí v rámci vhodného integračního a vyhledávacího systému.
3. Navrhněte systém pro detekci anomálií, který bude založen na umělé inteligenci, dle instrukcí vedoucího práce.
4. Po konzultaci s vedoucím práce navržený systém implementujte.
5. Implementovaný systém ověřte na vhodně zvolených reálných datech.
6. Diskutujte získané výsledky a možnosti dalšího rozšíření.

Literatura:

- Kurose, J. F. Computer networking: A top-down approach. Pearson, Essex, 2017, ISBN 978-1-292-15359-9.
- Stallings, W. Network security essentials: Applications and standards. Hoboken, 2016, ISBN 978-0-13-452733-8.
- Bishop, M. Computer security: Art & Science. Addison-Wesley, Boston, 2003, ISBN 0-201-44099-7.
- Ahmed, M., Mahmood Naser, A., Hu, J. A survey of network anomaly detection techniques. Journal of network and computer applications. Elsevier, 2016, 60(C), s. 19-31. ISSN 1084-8045.
- Buczak, A., Guven, E.. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. IEEE Communications surveys and tutorials. IEEE, 2016, 18(2), s. 1153-1176.

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

| | |
|---|---|
| Vedoucí práce: | **Očenášek Pavel, Mgr. Ing., Ph.D.** |
| Vedoucí ústavu: | Kolář Dušan, doc. Dr. Ing. |
| Datum zadání: | 01.11.2022 |
| Termín pro odevzdání: | 10.05.2023 |
| Datum schválení: | 28.10.2022 |

## Abstract

The objective of this work is to implement an anomaly detection system using artificial intelligence techniques that can detect anomalies by learning the system behavior. The proposed approach is effective in identifying novel or unknown anomalies that traditional rule-based methods may miss in network traffic data. However, the implementation of such a system involves addressing challenges such as data processing and feature extraction. This work discusses different methods of data analysis and intrusion detection approaches in Extended Detection and Response systems and the challenges we face in today's expanding security technologies.

## Abstrakt

Cílem této práce je implementovat systém detekce anomálií využívající techniky umělé inteligence, který dokáže detekovat anomálie učením chování systému. Navrhovaný přístup je účinný při identifikaci nových nebo neznámých anomálií, které tradiční metody založené na pravidlech mohou postrádat v datech síťového provozu. Implementace takového systému však zahrnuje i řešení problémů, jako je zpracování dat a extrakce charakteristických rysů. Tato práce pojednává o různých metodách analýzy dat a přístupech k odhalení průniků v systémech Extended Detection and Response a výzvách, kterým čelíme v dnešních rozšiřujících se bezpečnostních technologiích.

## Keywords

anomaly, detection, prevention, XDR, IDS, networks, endpoints, neural networks

## Klíčová slova

anomálie, detekce, prevence, XDR, IDS, sítě, koncové body, neuronové sítě

## Reference

POPOSKI, Vasil. *Anomaly recognition of extended detection systems.* Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Mgr. Ing. Pavel Očenášek, Ph.D.

# Anomaly recognition of extended detection systems

## Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Mgr. Ing. Pavla Očenáška, Ph.D Další informace mi poskytl pan Ing. Petr Chmelař. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

. . . . . . . . . . . . . . . . . . . . . . .
Vasil Poposki
May 9, 2023

## Acknowledgements

I express my gratitude to Mgr. Ing. Pavel Očenášek for his expert guidance in this work and Ing. Petr Chmelař for consultation on this project and valuable assistance.

# Contents

# List of Figures

# Chapter 1

# Introduction

With the amount of information roaming the network and the ever-increasing number of users, the vulnerability of various areas of the system infrastructure has increased. Extended Detection and Response (XDR) aims to provide a unified solution to this problem by incorporating known cybersecurity methods and techniques.

Gartner [13] describes XDR as “a SaaS-based, vendor-specific, security threat detection and incident response tool that natively integrates multiple security products into a cohesive security operations system that unifies all licensed security components”. The main goal of XDR systems is to provide an enhanced security layer and improve overall detection, protection, and response capabilities.

Cybersecurity tools have evolved over time due to the growth of the Internet and the fact that attacks became more sophisticated. Antivirus software was able to detect and remove malware from individual computers, while a firewall was introduced to protect the network from external threats. Intrusion Detection Systems and Intrusion Prevention Systems were designed to detect anomalies and protect the network using various advanced techniques. With the rising popularity of Cloud Computing, the Endpoint Detection and Response (EDR) approach was introduced to protect individual endpoints like laptops and mobile phones.

XDR collects and analyzes data from the entire infrastructure, including endpoints, networks, servers, emails, and the cloud. Thus, it enables event management, threat detection, and incident response in a single environment [6]. By improving **visibility** and **alert correlation**, we are able to better understand threats and enable faster actions against them.

# Chapter 2

# Anomaly analysis in XDR systems

In this chapter, I discuss the basic methods of anomaly analysis in XDR systems. Since anomaly analysis is a broad term, I focus on explaining the most commonly used approaches and methods that are proven to be the most effective in dealing with anomalies.

## 2.1 Intrusion detection system

The intrusion detection mechanism is a crucial component of every XDR system. If XDR is able to detect intrusion quickly enough, attackers can be identified, therefore any risk of damage will be eliminated, or at least the damage will be reduced. **Intrusion detection system** (IDS) and **Intrusion prevention system** (IPS) are both designed to secure systems from outside threats and provide protection to the network and individual users. IDS systems can detect malicious activity and provide alerts, while IPS systems can take action to prevent malicious activity. The process of anomaly detection is shown in Figure 2.5.

### Intruders and Threats

An intruder is an attacker who tries to gain access to a system and exploit system weaknesses. Stallings [21] divides intruders into three categories and gives us examples of typical intruder behavior patterns:

1. Hacker – Usually hacks into the computer to gain status in the hacking community. Can be benign or malign.
2. Criminal Enterprise – An organization or group of hackers with a specific target they aim to attack.
3. Insider – An employee using their permissions and knowledge of company systems to gain valuable information.

### Malicious software

Malicious software is any software that can harm or damage computer systems. Malicious software can exploit computer system weaknesses in a variety of ways. Some techniques of intrusion are far simple and do not require complex intrusion detection systems in order to detect or prevent them, while other more sophisticated intrusions require distributed systems that can detect or prevent a large scale of possible attacks.

There are several types of malware, including backdoors, Trojan horses, viruses, and worms. Viruses and worms are used to spread across networks while exploiting vulnerable parts of the infrastructure. Other types of malware include the exploitation of authentication services in order to get access to a specific system. A backdoor is a sort of malware that circumvents standard authentication processes to grant unauthorized access to a computer system, while Trojan Horses pretend to be trustworthy software while actually performing malicious actions, such as stealing private information or granting the attacker illegal access [21].

Distributed Denial of Service (DDoS) attack is a type of attack in which a large number of systems are used to flood a targeted system with traffic, making it unavailable to legitimate users. In Man-in-the-middle (MitM) types of attacks, an attacker intercepts communication between two parties to steal data, such as login credentials, credit card numbers, or other sensitive information. Cross-site scripting (XSS) attacks are a type of injections that occur in web applications. Vulnerable web applications allow an attacker to inject malicious code into a website and exploit its weaknesses. These attacks can occur when a website accepts user input, such as in a search box or comment section, without properly validating the input. In a similar manner, SQL injection is used to inject malicious code through a web application form. By inserting specific SQL statements, it can trick the application into executing the code.

There is a large scale of possible threats that may occur in the vulnerable parts of network infrastructure, applications, or on endpoints. The main countermeasure against these threats is detection and prevention.

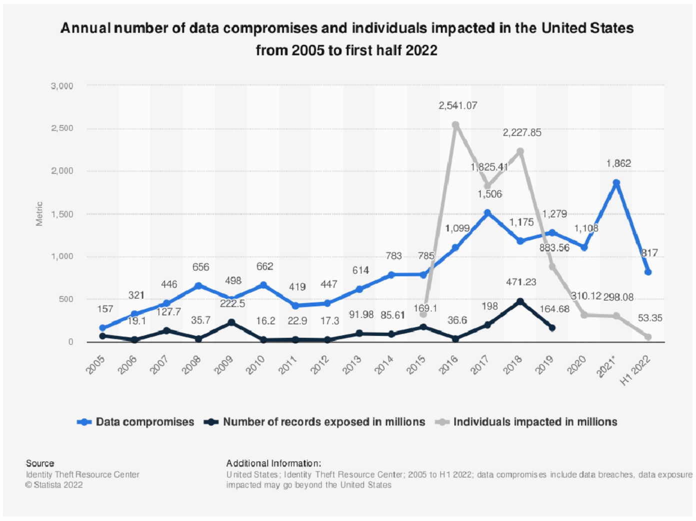

Figure 2.1: Cyber crime statistics 2005–2022 [7].

## 2.1.1 Network-based Intrusion Detection Systems

Network-based Intrusion Detection Systems (NIDS) are designed to detect malicious activities in network data. It works by constantly monitoring the network traffic and analyzing incoming packets. NIDS are typically deployed at strategic points in a network, such as at the perimeter or on specific critical servers. It's necessary for these systems to inform the administrator about suspicious activity and take countermeasures in order to block suspicious traffic and secure the network infrastructure.

Some common examples of attacks that NIDS can detect are Remote Command Execution (RCE) attacks, Denial of Service (DOS), Man-in-the-middle (MITM), Port Scans, and various malware infections such are worms, viruses, and Trojans. For example, NIDS can detect a series of TCP connections to different ports during a short period of time and flag this activity as a possible port scan.

### 2.1.2 Host-based Intrusion Detection Systems

The purpose of Host-based Intrusion Detection Systems (HIDS) is to protect individual devices, i.e., endpoints, such as servers, PC, or mobile devices. HIDS monitors endpoints and collects data from various sources like system logs, event logs, system calls, or performance data. To indicate the security breach, HIDS creates the baseline of normal behavior and analyzes collected data to identify suspicious activities.

A more advanced form of HIDS is **Endpoint Detection and Response** (EDR), which uses a more proactive approach to collecting and analyzing data. EDR also allows the remediation and investigation of the endpoint. **Wazuh**[1] and **OSSIM**[2] (Open Source Security Information Management) are two popular security solution used for collecting and analyzing data on endpoints. Besides real-time protection and incident response capabilities, they also provide logging and event management.

## 2.2 Rule-Based Analysis

This approach uses predefined signatures to detect malware. Signatures may contain various information such as IP address, port number, or a specific string found in the packet payload. If a certain pattern is found in incoming traffic, it may indicate malicious activity. Rule-based detection does not generate a large number of false alarms, due to the fact that rules are defined based on previous intrusions. However, this approach is not so effective in detecting new anomalies for which there are still no rules defined.

**Snort**[3] is a popular open-source intrusion detection system that uses a pattern-matching mechanism. It is based on a simple language for specifying rules [12]. Rules are written in a specific syntax and are organized into categories such as `alert`, `drop`, and `pass`.

**Suricata**[4] is a newer intrusion detection system that uses a multi-threaded architecture that allows it to process packets in parallel. This means that more data is processed, making Suricata a more efficient tool compared to Snort [22]. It is often used in conjunction with SIEM (Security Information and Event Management) tools to enhance the analysis process.

Rule-based intrusion detection tools have thousands of rules in the database, including a large number of rules contributed by the community [5]. Therefore, keeping rules up to date is an important part of maintaining network security.

#### Example of Snort/Suricata rule

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 22
(msg:"SSH login attempt"; sid:10000001; rev:1;)
```

[1]https://wazuh.com/
[2]https://cybersecurity.att.com/products/ossim
[3]https://www.snort.org/
[4]https://suricata.io/

This rule will trigger an alert if it detects any TCP traffic going to port 22, which is the default port for SSH.

## 2.3 Statistical Analysis

Statistical analysis is used to identify deviations from normal behavior. The term normal behavior here is used to describe legitimate or expected behavior. In terms of detection efficiency, statistical analysis is more efficient than rule-based analysis in dealing with intruders, which are unlikely to mimic normal behavior. Statistical analysis methods can be very useful in detecting previously unknown threats.

We can categorize statistical analysis based on outlier detection techniques into two categories [21].

- Threshold detection: The number of occurrences of a particular event is examined over a certain period of time. If this number exceeds the threshold we defined, we can assume the presence of outliers in the data. Since it is necessary to define a time interval and a threshold value for each event, in larger systems this analysis technique can produce a large number of false positives and false negatives.
- Profile-based detection: In order to detect anomalous activity, we create the profile based on previous user behavior. Normal user behavior is defined by a set of metrics. The detection mechanism compares the created profile with the user's current activities and determines the degree of deviation.

An example of a statistical analysis method is the Gaussian distribution model shown in Figure 2.2. This method assumes that the data being analyzed follows a normal distribution and calculates the probability of a given data instance being anomaly-based. We detect outliers by determining all data instances that are more than $3\sigma$ distance away from the mean $\mu$, where $\sigma$ is the value of the standard deviation of the dataset. About 99.7% of values are within the area of $\mu + -3\sigma$.

Another method to determine anomalies in operating system data is the $\chi^2$ statistical test. The output of a test is a measure of the deviation between the observed and expected values:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}, \tag{2.1}$$

where $O_i$ is the observed value and $E_i$ the expected value of the $i_{th}$ variable [8].

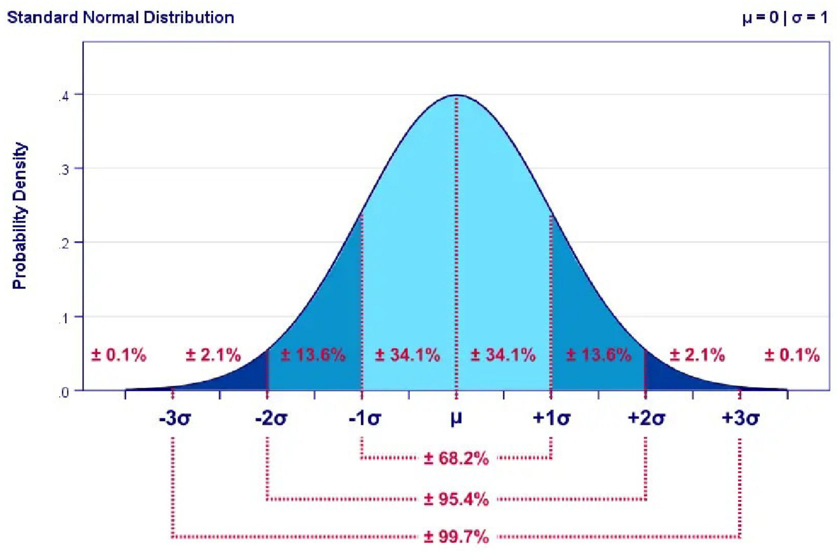

Figure 2.2: The area under the curve represents the probability for any interval of value [4].

## 2.4 Machine Learning-Based Methods

Machine learning techniques have become an important part when it comes to anomaly analysis. They typically involve training a model on normal behavior data and then using that model to identify deviation from the norm in the new data. In this subsection, some of the most commonly used machine learning techniques for anomaly detection will be discussed.

### *K*-means

$K$-means is a clustering-based method that can be used to group similar data points (instances) together. The algorithm first initializes $k$ centroids, where $k$ is a user-defined parameter, and assigns the data points to their nearest centroid. The algorithm then computes new centroids by taking a mean of each defined cluster and repeats this process until clusters no longer change or a stopping criterion is met. By grouping similar points into clusters we are able to identify clusters that deviate from the norm i.e. the outliers (Figure 2.3).

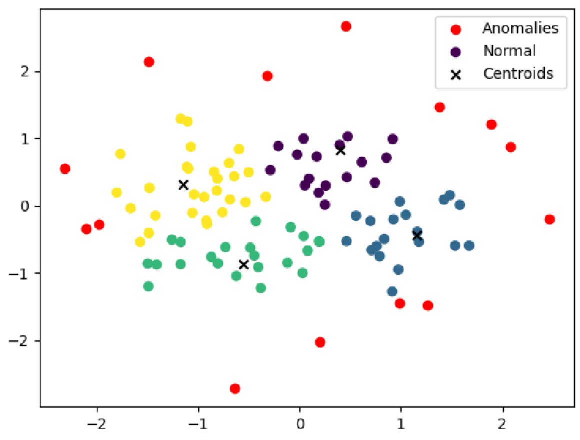

Figure 2.3: Example of clusters computed on random data using the $k$-means algorithm from the `scikit-learn` Python library with a predefined number of clusters and an anomaly threshold value.

### Decision Trees

A decision tree algorithm is a supervised algorithm used for classification. Decision trees consist of nodes, leaves, and edges. The algorithm works by starting at the root node and following the edges to the appropriate child node based on the input features. This process is repeated until a final prediction is reached at a leaf node.

### Artificial Neural Networks

Artificial Neural Networks (ANN) are computational models that are inspired by the structure of biological neurons in the brain. Artificial neurons are organized into layers that have connections with other layers. Artificial Neural Networks are often used in anomalous pattern identification. By using reinforcement learning, ANN is able to learn new types of attacks efficiently [5]. In this approach, the neural network is trained using normal data and then used to identify anomalous behavior in new data. In Figure 2.4 the general architecture of ANN is presented.

The algorithm for neural networks can be divided into two steps [8]:

- Training: A neural network is trained on a large dataset containing normal patterns of network behavior to classify and identify normal patterns. Using *backpropagation*, the weights of individual neurons are adjusted to improve the output results.
- Testing: Test instances of unseen data are provided as input to the neural network. The neural network then accepts or rejects the input data, based on what it learned in the training process. In case of rejection of input data, the instance is classified as an anomaly.

Each neuron in an ANN has an associated bias and weight. The bias is a constant that allows the neuron to adjust its output independently of its inputs. A neuron's weight is a quantity that measures how strongly its inputs and outputs are connected. To determine the neuron's output, the weighted sum of a neuron's inputs is first calculated. A bias value is then added, and the result is passed to an activation function. The activation function enables the neural network to represent complex interactions between inputs and outputs and therefore learn complex patterns in the data. The output of a neuron is passed as an input to the neurons in the next layer. Weights and biases are adjusted in the process of backpropagation. In this process weights and biases are learned values in order to minimize the loss.

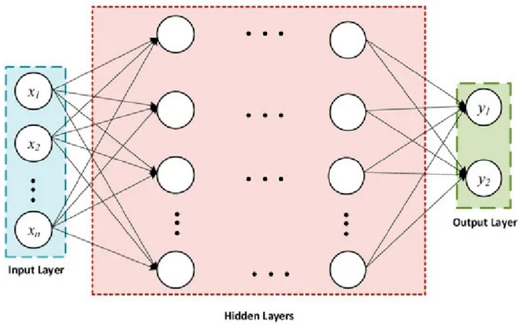

Figure 2.4: The general architecture of Artificial Neural Network [1].

## 2.5 Output of Anomaly Detection

The important part of the anomaly analysis is the output of the anomaly detection process (see Figure 2.5). Typically, the anomaly detection result can be reported in two ways [8]:

1. **Scores**: Scores are assigned to each anomaly detection output instance. Typically, the higher the score, the higher the probability that this particular instance is truly an anomaly. Using this technique, the administrator is able to see all potential anomalies and manually mark some as anomalies, or can automatically select anomalies by defining a threshold value.

2. **Labels**: Output instances are marked binary, i.e. using the labels - *normal* or *anomalous*.

| **Instance** | **Score** | **Label** |
|---|---|---|
| A | 0.4 | False |
| B | 0.78 | True |
| C | 0.2 | False |
| D | 0.35 | False |
| E | 0.9 | True |

Table 2.1: Example of instance output using scoring and labeling.

The anomaly score is calculated by a set of metrics and its estimation can be formulated using several techniques, including *distance-based*, *density based* and *soft computing based* technique [5].

## 2.6 Evaluation of Anomaly Detection

Another important aspect is the evaluation of the output. During this process, the effectiveness of the output results is determined. Using the appropriate evaluation measure we can evaluate the precision of intrusion detection.

Evaluation typically involves comparing output results with normal or expected behavior and measuring false alarm rates. *Precision*, *Recall* and *F-score* are commonly used measures [5].

Evaluation is used to help the system achieve better performance in detecting and preventing malicious activities and optimize the process of anomaly analysis.

Evaluation is an important phase of the implementation of the detection system, which allows us to gain insight and be able to see how well the anomaly detection system works on different data. Based on this information, the system can be further adjusted and changes can be made to the system design to maximize the anomaly detection rate.

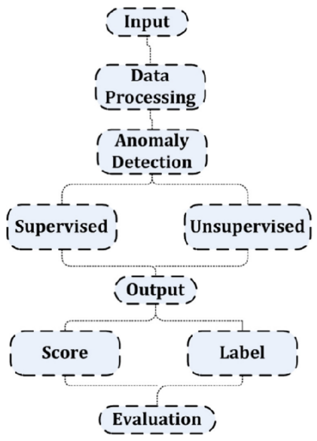

Figure 2.5: The generic process of anomaly analysis [2].

# Chapter 3

# Anomaly Detection System Design

In this chapter, I will be discussing system requirements to enable the analysis of data from the Network-based Intrusion Prevention Systems (NIPS) and Host-based Intrusion Prevention Systems (HIPS) with an open license within the appropriate integration and search system. I will then discuss the initial design of the system I will be implementing.

## 3.1 Requirements

In order to maximize the efficiency of anomaly analysis, the main requirement is to have an IDS/IPS that can recognize both known and unknown anomalies. A hacker with the intention to breach the system security is able to launch an attack using tools like HOIC and LOIC [5]. These tools are capable of many sophisticated attacks that can harm security systems and cause problems. That's why the security system has to cover all important areas of infrastructure. The second major problem with today's cybersecurity technologies is a large number of false alarms. This can be reduced by using an efficient anomaly analysis method, training the model appropriately, and constantly keeping track of new potential threats that may occur.

There are several key requirements that need to be considered here:

- **Data collection**: The system should be able to collect data from various sources. This includes network traffic, endpoints, and servers. Data collection provides visibility and insight into the most vulnerable parts of the system. This often includes data mirroring and real-time monitoring (data capturing). Data is collected using tools that are capable of accessing and inspecting specific data sources and that are able to parse this data into a specific format.

- **Data storage**: To be able to handle a large amount of incoming data, we need flexible data storage. Network and endpoint data must be stored and retrieved efficiently for analysis purposes. Tools that can be used for this purpose are **Elasticsearch** and **Apache Kafka**. Apache Kafka is used for data processing as well.

- **Feature extraction**: Relevant data has to be extracted from various sources and structured into a format that is suitable for anomaly analysis. This step also includes the preparation of data for analysis and normalization. In order to extract and normalize features, various tools can be employed. Python is a well-known language used by machine learning analysts, which provides libraries for data structuring, formatting, and normalization. Some of the most common tools used in Python for machine learning analysis are Skit-learn, Pandas, and Numpy.

- **Data analysis**: Data analysis requires an intrusion detection mechanism using an appropriate detection method. This step requires the implementation of the intrusion detection system integrated with existing open license tools. A deep learning model will be employed for this purpose since it can provide profound analysis and detection of complex patterns in large datasets. Open-source libraries such as TensorFlow and Keras provide access to general APIs for building machine-learning model architecture and training. Once the model is trained, it can be used to classify real-time network traffic and identify potential intrusions.
- **Data visualization**: The system should include a simple visualization interface for security events analysis. There are multiple existing tools that can be used for this purpose, e.g. **Kibana**, **Grafana**, and **Squert**.

## 3.2 Implementation design

For the purpose of this bachelor thesis assignment, I designed the Anomaly Detection system, which uses a machine learning approach to learn the normal behavior of the system. The system will be implemented in Python language using appropriate available libraries. The machine learning model will be based on Artificial Neural Networks and trained on normal behavior datasets. To test the model's efficiency in detecting anomalous behavior, several tools and methods will be employed. The performance of the implemented system will be evaluated using various classification evaluation techniques.

For the purpose of efficient anomaly analysis, the system will be integrated together with Suricata, an IDS/IPS tool that contains rules for a large number of possible attacks and is capable of effective monitoring both on the network and endpoints.

For the purpose of data storage and visualization, two open-source **ELK**[1] products will be integrated: Elasticsearch and Kibana.

Implementation will be focused on detecting different varieties of DoS and brute-force attacks. The anomaly detection system will be tested on preprocessed data and by using suitable penetration testing tools.

[1]https://www.elastic.co

# Chapter 4

# Anomaly Detection System Implementation

In this chapter, I will be discussing the implementation of an Anomaly Detection System as the main part of my work on my bachelor's thesis. I will focus on explaining the key aspects of building an anomaly detection system, such as the implementation environment, tools and techniques, and the data analysis process. I will discuss the process of preparing data for analysis in several stages, as well as the process of implementing a neural network that will be used to detect anomalous behavior. Finally, various testing tools and known evaluation methods utilized in anomaly detection system performance assessments will be employed to examine the implemented system performance.

## 4.1 Environment

To begin, selecting the appropriate implementation environment is crucial for the successful implementation of an anomaly detection system. This involves choosing tools and programs that can handle large volumes of data efficiently. Anomaly Detection System is built within Security Onion, a free and open-source security software used for anomaly detection and prevention. Security Onion integrates a number of security tools such as Suricata, Snort, Zeek, Wazuh, and Osquery among others. It provides a comprehensive monitoring platform that is used by a large number of security teams.

### 4.1.1 Environment Setup

Security Onion is installed using Security Onion ISO image, which is based on CentOS 7 and Oracle's virtualization software VirtualBox. Security Onion offers multiple installation options, including multiple types of deployment. The architecture that was most suitable for the purpose of efficient implementation and testing of the Anomaly Detection System was *Standalone* architecture shown in 4.1.

The basic idea behind *Standalone* architecture is the deployment of all components on a single machine. In this mode, Filebeat, a log shipping tool, is responsible for transferring log files and other files to Logstash, another data processing pipeline tool that is used for collecting, processing, and forwarding logs to a centralized location for analysis. Logstash uses two pipelines for data transfer. The first pipeline is used for collecting data from Filebeat and transferring it to Redis for queuing. The second pipeline extracts data from Redis and sends them to Elasticsearch for analysis.

**Network Configuration**

An important aspect of environment setup is a network interface controller (NIC) setup. Security Onion Standalone requires two network interfaces:

1. Management interface – Interface used for accessing Security Onion Console and overall system management. This interface is assigned an IP address using DHCP.
2. Sniffing interface – Interface used for monitoring and capturing network traffic. By allowing promiscuous mode on this interface in VirtualBox, all network traffic is passed through the physical interface and processed by the virtual interface.

Both interfaces are set to *Bridged*. They are linked to the host machine's physical network. This duplicates a network node and allows the virtual machine to function as part of the same network as the host machine [5]. It enables the virtual machine to receive and send network traffic just like a physical computer would on the network, making network traffic capture and analysis for security monitoring and analysis simpler.

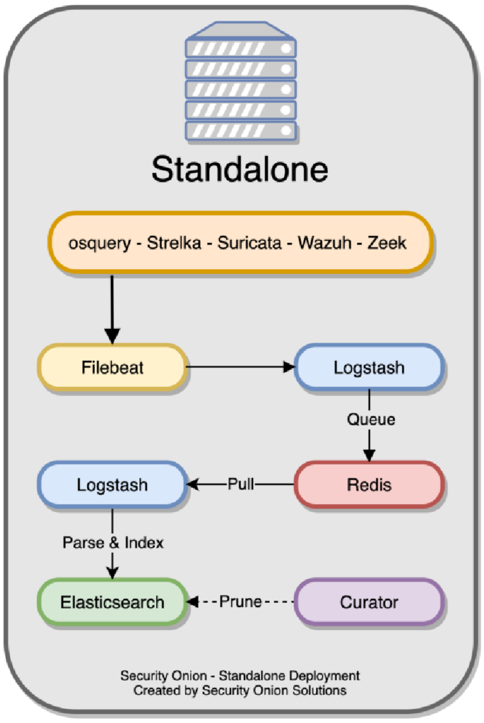
Standalone
osquery - Strelka - Suricata - Wazuh - Zeek
Filebeat
Logstash
Queue
Logstash
Pull
Redis
Parse & Index
Elasticsearch
Prune
Curator
Security Onion - Standalone Deployment
Created by Security Onion Solutions

Figure 4.1: Security Onion Standalone architecture [20].

### 4.1.2 Environment Tools

Some of the main tools included in Security Onion and used for implementing an Anomaly Detection System are [20]:

- **Suricata**: The role of the Suricata Intrusion Detection System (IDS) in Security Onion is real-time traffic analysis and detection of various types of network attacks,

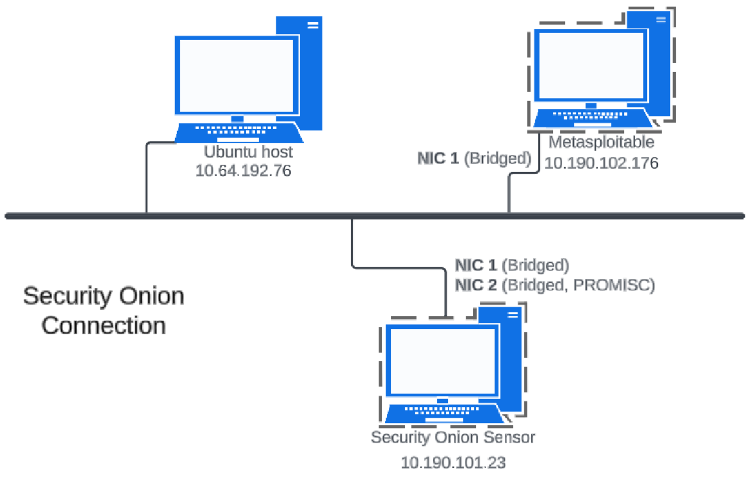

Figure 4.2: The setup used during anomaly detection system implementation.

including malware, exploits, and brute-force attacks. Suricata generates alerts while monitoring network traffic using signature-based detection, i.e. rules that are applied to incoming traffic to detect known threats and malicious activity. When Suricata detects a signature match, it generates an alert that can be found immediately in *Alerts* interface in the Security Onion Console (SOC) or be visualized using various visualization tools available in Security Onion like Kibana and Grafana.

- **Osquery**: Osquery is a tool used in Security Onion for system inventory and event monitoring. It enables analysts to collect and analyze system-level data from Security Onion sensors and other endpoints.
- **Elasticsearch**: Elasticsearch is a distributed, RESTful search and analytics engine used for log management and analysis in Security Onion. It is used to store and index log data and enables fast and efficient searching and analysis of large data sets. Index log data is collected using various anomaly detection tools.
- **Salt**: Salt is a configuration management and orchestration tool for managing Security Onion sensors and other components. It enables centralized management and configuration.
- **Kibana**: The primary visualization tool that was used for testing purposes during the implementation of the Anomaly Detection System. Kibana is a data visualization and exploration platform used to analyze and visualize large data sets. It enables users to create custom dashboards and charts to gain insights and visibility into log and performance data. It relies on Elasticsearch API to successfully retrieve data for analysis.

The tools in Security Onion were tested for functionality using publicly available PCAP (Packet Capture) files. PCAP files were replayed to the sniffing interface of the Security Onion sensor using the tool developed by Security Onion called `so-import-pcap`. Using this network traffic replaying tool, we can evaluate the functionality of Security Onion components in the ability to analyze traffic and detect malicious activities. The main advantage of this tool is to reproduce the network traffic while preserving the original timestamps of the packets. This allows the isolation of specific packets or flows from the rest of the traffic and facilitates analysis.

By default, the Security Onion sensor blocks all incoming traffic to ensure maximum security. Another useful command line utility in Security Onion is `so-allow`, which allows the unlocking of the firewall to connect to Security Onion from new IP addresses. This utility is specifically used for enabling access to the Security Onion Console and the Elasticsearch REST API.

## 4.2 Data preparation

A sizable amount of data must be acquired and thoroughly processed to construct the deep learning model. To train deep learning models, we require data that is stable and adequately represented. Multiple data preparation aspects can leverage the deep learning model performance and its ability to detect anomalous behavior [16]. One important aspect is the data structure, which refers to the format and organization of the data. The data must be structured to be compliant with the model's architecture and requirements, including maintaining a consistent size and format. Another important aspect is data stability, which refers to the consistency of the data over time. The model needs to be trained on stable data that represents the normal behavior of the system, to accurately detect anomalous behavior. The availability of anomaly data is also crucial for improving the performance of the model. Since the deep learning model is trained only with normal behavior data, we need anomaly data to be able to test the model and evaluate its performance. Anomaly data should represent a variety of potential threats.

For the model to be trained to detect a variety of distinct anomalies, data diversity is crucial. The training data should comprise several various types of typical behavior. This typically includes a variety of network communication protocols and different communication types.

Next, I will explain the two main stages of data preparation for training a deep learning model.

### 4.2.1 Data collection

Suricata serves as the primary network data collector in the Security Onion environment. Once the data is collected, it is transferred using Filebeat and Logstash and stored on the Elasticsearch server. For all types of events that occur in the network, Suricata generates a variety of EVE JSON logs. These logs include data on network traffic, packet captures, and specific protocol information. Elasticsearch stores data in the form of documents, which are then stored in the form of indices. Each document can have an unlimited number of fields with each field containing a value or an array of values. In addition to the NoSQL storage mechanism, Elasticsearch has an effective search engine that enables fast data retrieval using REST API [14].

The general structure of the EVE JSON log consists of several sections which provide general information about specific events, such as `timestamp`, `event`, and `network`. Sections `source` and `destination` provide information about source and destination IP addresses and port numbers, as well as geographical information about specific endpoints. Events are categorized into multiple *datasets*, where each dataset corresponds to the network protocol used in collected traffic. Based on the dataset field value, each log contains an additional section that provides information related to a specific dataset (protocol).

Data stored on the Elasticsearch server is collected using the Python Elasticsearch client from the `elasticsearch` library. The Elasticsearch client allows flexibility in accessing the Elasticsearch RESP API and collecting logs for further processing.

### 4.2.2 Feature extraction and data reformatting

To facilitate the process of collecting EVE JSON logs, as well as parsing and structuring, a special API has been implemented. The `FeatureExtraction` class is responsible for both collecting data from the Elasticsearch server using queries and parsing the data into a format suitable for deep learning analysis.

Based on the data preparation requirements discussed earlier, each collected log was parsed, and essential features were extracted. Python's built-in `dataclasses` module was utilized to hold general information obtained from the logs, as well as properties that are unique to each dataset of a given log. Once all the logs were parsed, and features were extracted from them, the resulting data was initially reformated using the `DataFrame` from `pandas` library. This library is capable of representing data in a tabular format consisting of rows and columns, with each column capable of containing different data types. For data preprocessing and analysis in machine learning and deep learning, Pandas and dataframes are frequently utilized, since they offer convenient methods and techniques for managing large datasets and performing data transformation.

The final stage of data preparation rests on transforming all columns into numerical data types and encoding individual columns. Removing any irrelevant information also helps to reduce noise and improve the accuracy of the model. IP address values are categorized based on the class the IP address belongs to, while port numbers are categorized according to known port number ranges.

In addition to encoding numerical data types, the transformation process also involves converting categorical data into numerical representations through one-hot encoding. One-hot encoding is one of the most common encoding techniques to handle categorical data. Its advantage compared to other encoding methods is its ability to handle non-ordinal features, by creating a binary vector. Binary vector will have a length of $n$ where $i$-th element is 1 if the data corresponds to the $i$-th element and 0 otherwise [16].

| `protocol` |
|---|
| http |
| ftp |
| ssh |

$\longrightarrow$

| `protocol_http` | `protocol_ftp` | `protocol_ssh` |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

Table 4.1: Example of one-hot encoding of a variable.

Normalization of categorical features is done using Min-Max scaling. The Min-Max normalization method transforms numerical features by scaling them into the range $[0, 1]$, where 0 will be the minimum possible value for a given feature and 1 the maximum value. This is usually done to guarantee that each feature, regardless of its initial scale, contributes equally to the analysis. For this purpose, `MinMaxScaler` available in `skit-learn` library has been used. The formula for min-max scaling is:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}, \tag{4.1}$$

where $x$ is the given feature value, $min(x)$ is the minimum value of the feature, $max(x)$ is the maximum value of the feature, and $x'$ is the final, normalized value.

The final dataframe has a shape of $(58908, 139)$, where the first element represents the number of indices (rows) in the dataframe and the second element represents the total number of features (columns). Out of 139 features, 134 features represent one-hot encoded categorical columns. The remaining 5 features are numerical features normalized using min-max scaling. All of the categorical features are now represented using `numpy.uint8` data type, while numerical features are represented using `numpy.float64` data type.

## 4.3 Neural network implementation

In this section, I will discuss neural network model implementation, as well as a choice of neural network model architecture. The process of training the neural network model and parameter tuning will also be explained in detail.

### 4.3.1 TensorFlow framework

TensorFlow is an open-source machine learning framework that offers a wide range of tools and APIs for implementing models. It is used in a variety of fields, including natural language processing, image, and speech recognition, as well as anomaly detection. TensorFlow provides a high-level API for building and training neural networks.

One of the key aspects of TensorFlow computation ability is a computational graph. The computational graph is a way of representing a computation as a graph, where graph nodes represent computing operations. Edges of a computational graph define the data (tensor) flows between the nodes in a directed manner [15].

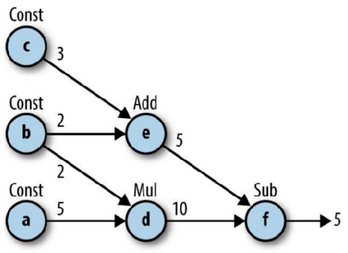

Figure 4.3: Illustration of a simple computational graph in TensorFlow with defined operations and data flows [15].

### 4.3.2 Autoencoder

To implement a neural network-based anomaly detection system, I chose the autoencoder neural network. Autoencoder is a neural network model that aims to reconstruct input data. The main task of autoencoder models is to reduce the dimensionality of input data in the encoding stage and attempt to reconstruct data in the decoding stage. According to [10], autoencoder demonstrates effectiveness in anomaly detection, particularly in resilience against noisy log training data. In their experiment, they tested the accuracy of several most common neural network models in anomaly detection. There are several types of Autoencoder that have been employed for the purpose of detecting anomalous behavior, besides the standard model, e.g. Variational Autoencoder (VA) that adds a probabilistic component to encoding stage and Conditional Variational Autoencoder (CVA) that adds conditional components in encoding or decoding stages based on the context, such as event type [16].

While compressing data into smaller dimensionality, the autoencoder aims to preserve as much information as possible, while also allowing complex nonlinear transformations. The optimizer (e.g. *Adam*, *SGD*) automatically adjusts the weights of the models to minimize a given loss function during training [15].

By reducing the size of each layer in the encoding stage, input data is represented by a smaller number of features. In different machine learning models, this approach is utilized with the aim of reducing noise from data and facilitating the learning process. The encoding process ends with *bottleneck*, a hidden layer that holds the smallest compressed representation of input data. During the decoding process, data is reconstructed from the compressed representation in the bottleneck layer back to its original size.

The determination of the number of layers in the encoding and decoding phases of the autoencoder model is based on the shape of the input data. Since the input dataframe contains both different types of features, i.e. 134 categorical features and 5 numerical features, the data is partitioned, and distinct autoencoder architectures are applied to each respective part.

For the encoding stage, the part that contains categorical features is passed through three dense layers, with each layer decreasing the size of the previous layer. A dense layer is a fully connected neural network layer, which means that all neurons in the preceding layer are connected to all neurons in the following layer. Starting from the input layer, which consists of 134 units, the size of subsequent layers is gradually reduced to 128, 64, 32, and finally 16 units (see Figure 4.5). The activation function used in the encoding layers and first two decoding layers is Rectified Linear Unit (ReLU) activation function. The ReLU activation function introduces non-linearity and enables a model to discover more complex data patterns. In the final layer of the decoder, the sigmoid activation function is used. A sigmoid activation function can effectively reconstruct the original categorical data because it outputs values in the range $[0, 1]$, which correspond to the range of values introduced in the encoder input layer.

In the case of numerical inputs, two dense layers are employed to compress the data into a smaller representation. This compressed representation is then used as input to another two dense layers that are used to decompress the data back to its original size. The number of neurons in the first dense layer is decreased from 5 units (the size of the input layer) to 3, and finally 2 units. The activation function used in the encoding layers is Rectified Linear Unit (ReLU), while the sigmoid activation function is used in the final layer of decoding numerical inputs.

To improve the performance of our model, the dropout regularization technique was employed. This technique is used to ensure that the neural network distributes the learned representations of the data across all the neurons by randomly deactivating a fraction of them during training. By placing dropout layers inside the encoder and decoder, the network is forced to learn a robust representation that can generalize well on new, unseen data. This regularization technique is primarily used to improve the model's generalization [15].

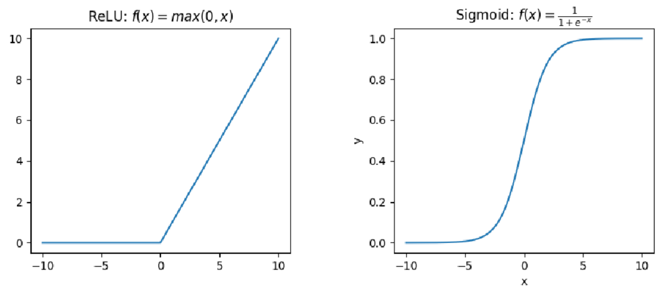

Figure 4.4: Plots of the activation functions used in neural network layers.

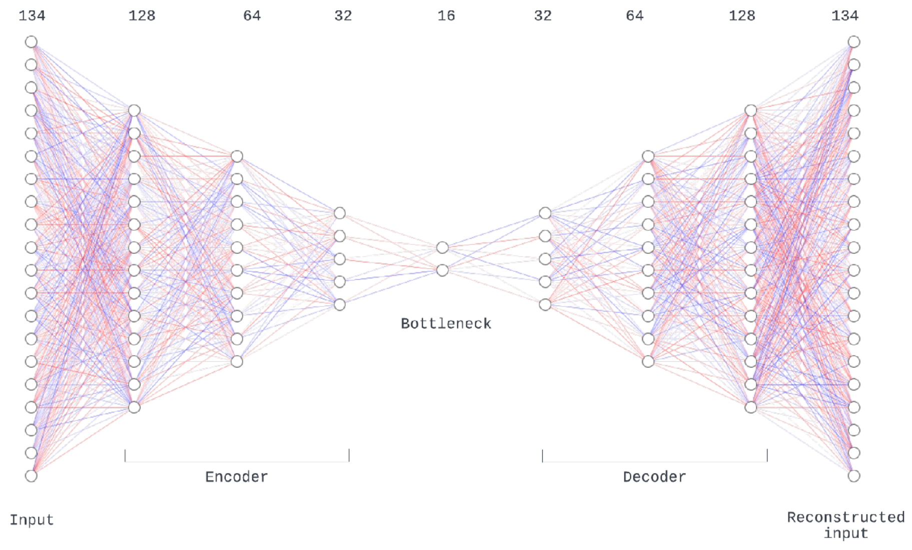

Figure 4.5: Autoencoder architecture used for categorical features.

### 4.3.3 Neural network training

The built neural network was trained on chosen normal network traffic logs that have been obtained over the course of two weeks using Suricata. Data was fetched from multiple Elasticsearch indices and was extracted and preprocessed using the implemented `FeatureExtraction` class.

The final output of the autoencoder is created by concatenating the reconstructed category and numerical features. The decoded layers are combined using the `concatenate()` method to merge the decoded numerical and category features. The final decoder layer has the same shape as the input data, which is the goal of the autoencoder – to reconstruct the input data as precisely as possible.

Before the model is trained, it is configured using the `compile()` method. The `compile()` method is used to configure the model for training. Here, the model is compiled using the mean squared error (MSE) loss function and the *Adam* optimizer. MSE measures the average squared difference between the predicted output and the actual output. The main purpose of the mean squared error loss function is to minimize the difference between the input and the output. The Adam optimizer is a stochastic gradient descent optimization algorithm that updates the learning rate adaptively based on the gradient of the loss function [19]. Additionally, the model is configured to calculate the accuracy metric during training.

Input data is divided into a training set and a validation set, for the purpose of training and validating implemented machine learning model. Using `sample()` function, the fraction which includes 80% of input dataframe is selected for training, while the remaining 20% of data is set for validation.

The model was trained for 35 epochs with a batch size of 16. The loss and accuracy metrics for both the training and validation datasets at each epoch are included in the `history` object that the `fit()` function returns, as shown in Figure 4.6. This object provides information about the training process. Using this data, it is possible to assess the model's effectiveness during the process of model implementation and adjust the model's architecture or hyperparameters.

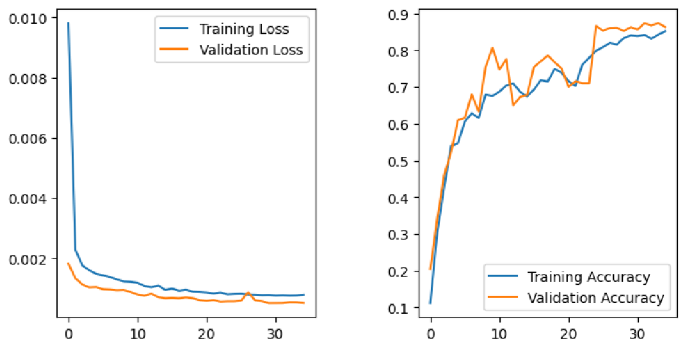

Figure 4.6: Loss and accuracy progress recorded during 35 epochs of neural network training.

# Chapter 5

# Experiments and Evaluation

In this section, I will outline the steps involved in testing and evaluating the neural network model that has been implemented. This process can be divided into three parts. In the first part, which includes defining a threshold for generating output predictions, the neural network is used on new non-anomalous data. In the second part, various experiments were conducted to see how well the neural network model performs on anomalous data. For this part, data from external sources is collected and various testing tools were utilized. In the final, third part, an evaluation of the neural network model is performed. The results of the evaluation are then analyzed to determine whether the model is performing well and meeting the desired criteria.

## 5.1 Prediction

To measure how well the neural network is able to reconstruct the original input data, output predictions are generated for the new input samples. Since the autoencoder is trained on normal, non-anomalous data, the level of how good the data reconstruction is will determine, if the input data is indeed normal, or is anomalous. Autoencoder was trained to minimize the reconstruction error and therefore improve the quality of decoding the encoded data. In theory, the autoencoder should reconstruct the normal data with a minimum loss, since it is trained to be able to successfully encode network traffic data and decode it back to its original size. Most of the anomalous data contains patterns that are not frequent in normal network traffic data, therefore autoencoder will have worse results in reconstructing this kind of data. This higher reconstruction error can indicate anomalous behavior and potential threat.

Output predictions are generated using `predict()` method, which takes input dataframe and computes predictions using the trained neural network model. Next, reconstruction error is calculated for each row in the predicted data. Predicted data is compared to the actual data to calculate the loss. The reconstruction error is a measure of how well the autoencoder is able to reconstruct the original input data. Reconstruction error is calculated by taking the mean square error between the predicted data and the actual data for each dataframe row. This is done by calculating the square of the difference between the predicted data and the actual data and then taking a mean of this value for each row. When the reconstruction error is lower, the model is performing better at recreating the input data, and when the reconstruction error is higher, the model is performing worse.

$$reconstruction\_error = \frac{1}{m}\sum_{i=1}^{m}(predicted\_data_i - data_i)^2 \tag{5.1}$$

### 5.1.1 Threshold

An important aspect of anomaly detection using an autoencoder is determining the optimal threshold value. The threshold value is used to classify data points with a reconstruction error above the threshold as anomalous and data points with a reconstruction error below the threshold as normal. There are several methods that can be used to define a threshold value. One of the most common methods includes calculating the mean and standard deviation of the reconstruction error. This method is defined as:

$$\text{threshold} = \bar{r} + k \cdot s \tag{5.2}$$

where $\bar{r}$ is a sample mean of the reconstruction error, $s$ is a standard deviation of the reconstruction error, and k is the standard deviation multiplier.

Another method for defining the threshold value is the percentile method. By calculating the $n$-th percentile of the reconstruction error, the reconstruction error value which lies above $n\%$ of the data points in the reconstruction error array will be used as a threshold value. In the process of testing the autoencoder performance, this method is used to determine the base value of a threshold.

For the purpose of defining the threshold value, normal network traffic data was extracted from the Elasticsearch index and analyzed using the trained neural network model. This data is new and has not been used during the neural network training process. Using the 99th percentile value of the reconstruction error array, a value of 0.03612 was obtained. Using this value, 99% of the input samples were classified as normal, which indicates the precision of 99% when analyzing new normal data using the autoencoder.

## 5.2 Replaying PCAP files

In this stage, the performance of neural network performance will be tested using the anomalous data samples, which were obtained by importing publicly available PCAP files. Packet capture files are obtained from [3], a website whose authors provide a large amount of packet capture files and malware samples for testing and analysis purposes. The effectiveness of the neural network model was tested in two experiments. The purpose of this evaluation is to ascertain how well the neural network can identify unusual network traffic. PCAP files were replayed on Security Onion virtual machine using Security Onion `so-import-pcap` utility. After they were analyzed by Suricata, logs were stored on a separate Elasticsearch index, with preserved timestamps. Using the Elasticsearch client, documents from a specific index were collected and processed. In order to extract the relevant feature from obtained data, `FeatureExtraction` class was used. Data is processed in a similar manner as the data used for neural network training, except for the additional data preparation step that was done in order to format data to suit autoencoder input dimensionality. In order for input data to match the expected input dimensionality of the autoencoder, missing columns were added to the input dataframe, and additional columns were removed.

To be able to validate the prediction for each log sample (normal/anomaly), output predictions were correlated with Suricata alerts. Validation of the neural network prediction requires labels. Using the combinations of source and destination IP addresses from *alert* dataset category, which was generated by Suricata, input samples were labeled using boolean value `True`, if a sample is anomalous and `False` if it is normal. When a certain rule is triggered, Suricata generates an *alert* log that contains information such as source/destination IP address, source/destination port, protocol, and more. This allows certain flows to be labeled as either anomalous or normal. The label array is removed from the input data with order preserved and saved for the post-prediction step.

After the prediction has been conducted, the True Positive Rate (TPR) and False Positive Rate (FPR) are calculated to evaluate the performance of the autoencoder. TPR measures the percentage of correctly identified positive samples (anomalies), while FPR is used to measure the percentage of samples that were incorrectly identified as anomalies.

### 5.2.1 Experiment 1

In the first experiment, PCAP file `2018-06-30-traffic-analysis-exercise.pcap` was replayed using `so-import-pcap` command. Data containing 1653 logs in total is stored in a separate Elasticsearch index. Data is extracted and prepared for neural network analysis. The Suricata rule *ET POLICY Data POST to an image file (gif)* was triggered when the PCAP file was replayed. Suricata identified malware in the analyzed data and categorized this event as *Network Trojan* with high severity. Based on the obtained alerts, labels were created for each sample and 9 rows in the input dataframe were categorized as anomalies. In the prediction phase, the reconstruction error array was obtained using a trained autoencoder.

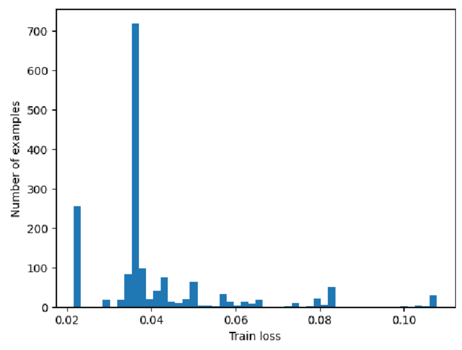

Figure 5.1: Experiment 1 – Reconstruction error plot.

Each sample in the reconstruction error array is compared to the threshold value calculated on a set of normal network traffic data earlier. Predictions for each sample in the reconstruction error array are represented as boolean values, where value `True` indicates that the reconstruction error is higher than a threshold and value `False` indicates that the reconstruction error is lower than a threshold. At this point, the predicted values can be compared with the previously created labels. The evaluation of prediction is based on the determination of True Positive Rate (TPR) and False Positive Rate (FPR). Using the defined threshold value, all anomalous samples were successfully identified. Some of the samples were also incorrectly identified as anomalies. Increasing the threshold value by 0.01 managed to reduce the FPR while maintaining the maximum TPR value (Figure 5.2), as shown in the table below.

| Threshold | TPR | FPR |
|---|---|---|
| 0.03612 | 1.0 | 0.3449 |
| 0.04612 | 1.0 | 0.1867 |

Table 5.1: Experiment 1 – True Positive Rate (TPR) and False Positive Rate (FPR).

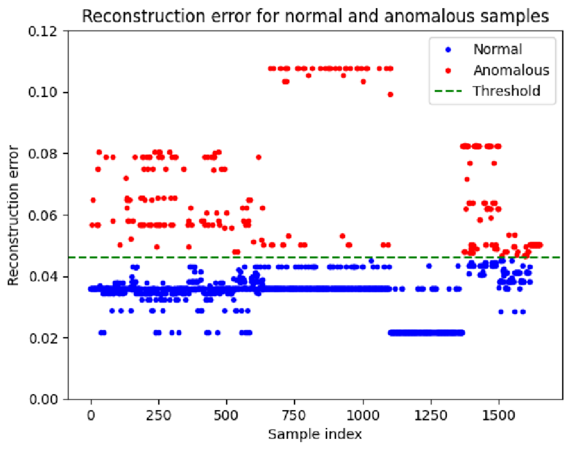

Figure 5.2: Experiment 1 – Classified prediction outputs.

### 5.2.2 Experiment 2

In the second experiment, PCAP file `2020-09-16-Qakbot-infection-traffic.pcap` was downloaded and replayed using `so-import-pcap` command. After it was analyzed by Suricata, logs were stored in the Elasticsearch index, 184 in total. Compared to the first replayed PCAP file, this dataset contains a far less number of samples. However, these PCAP files triggered a much larger number of rules:

- *ET JA3 Hash - [Abuse.ch] Possible Quakbot*
- *ET MALWARE Observed Qbot Style SSL Certificate*
- *ET INFO EXE - Served Attached HTTP*
- *ET JA3 Hash - [Abuse.ch] Possible Gozi*
- *ET MALWARE JS/WSF Downloader Dec 08 2016 M4*
- *ET MALWARE Likely Evil EXE download from MSXMLHTTP non-exe extension M2*
- *ET POLICY PE EXE or DLL Windows file download HTTP*

Suricata was able to successfully identify Quakbot malware. Quakbot (Qbot) is a common trojan-type malware that was to designed to monitor infected machines and misuse private and valuable information. Malware has been active since 2008 [9].

After the data is collected from the Elasticsearch index and prepared for neural network analysis, alerts were analyzed to identify all anomalous flows. In total 47 anomaly samples were identified in the test data using alerts and labels used for model evaluation were obtained. Using the defined threshold value, prediction for each sample is determined.

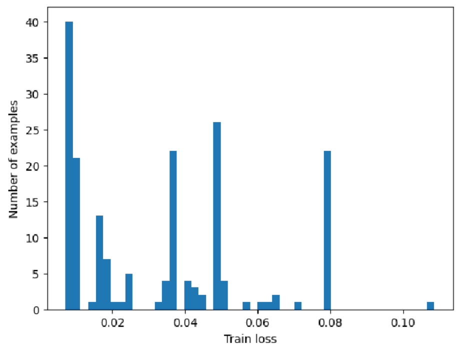

Figure 5.3: Experiment 2 – Reconstruction error plot.

Finally, the True Positive Rate and False Positive Rate were calculated to evaluate model performance. Calculated results were significantly worse compared to the first experiment. Defining a different threshold improved the results. In this case, a lower threshold value has better results in detecting anomalies but also increases the false positive rate, as shown in the table below.

| **Threshold** | **TPR** | **FPR** |
|---|---|---|
| 0.03612 | 0.04256 | 0.49635 |
| 0.01 | 0.97872 | 0.7153 |

Table 5.2: Experiment 2 – True Positive Rate (TPR) and False Positive Rate (FPR).

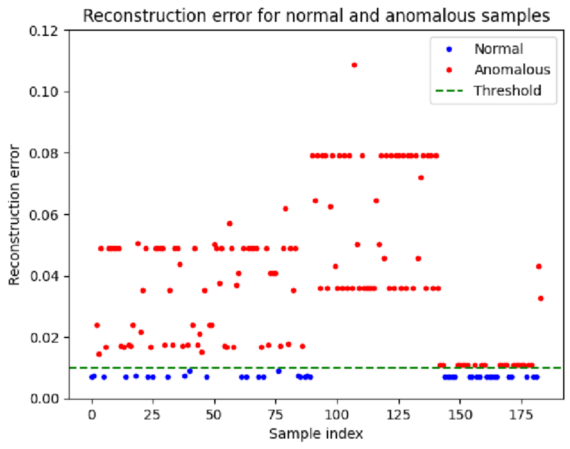

Figure 5.4: Experiment 2 – Classified prediction outputs.

## 5.3 Penetration testing

For the second part of the autoencoder performance evaluation, a special security environment is set up to test a real-life attack scenario. For this purpose, the Metasploit framework was installed on a local machine. Metasploit [17] is an open-source penetration testing platform that helps in identifying and exploiting vulnerabilities in computer systems and networks. It provides a large number of attacking tools that can be utilized for testing purposes.

### 5.3.1 Testing environment

This experiment aims to simulate an attack scenario that will exploit potential weaknesses. Metasploit provides *auxiliaries*, which are scripts that contain code for exploiting certain services.

The target of this attack will be a vulnerable virtual machine. Metasploitable is a Linux virtual machine, that is created for testing purposes. It contains several known vulnerabilities that can be efficiently exploited using penetration testing tools. This virtual machine has been installed on VirtualBox and connected to Security Onion virtual machine using `so-allow` command line utility. The traffic to and from the Metasploitable virtual machine is all visible to Security Onion. The environment is now ready to be used for penetration testing. In this experiment, three roles were defined:

- Metasploit: Attacker
- Metasploitable: Victim
- Security Onion: IDS

To simulate an attack scenario, the experiment used Metasploit as the attacker, Metasploitable as the victim, and Security Onion to monitor the traffic for suspicious activity. Data that was collected by Security Onion will be analyzed using the trained neural network model.

### 5.3.2 Attack launching experiment

Two brute-force attacks were conducted on Metasploitable virtual machine using the Metasploit framework. The primary goal of a brute-force attack is to gain control over a host or service. This attack typically involves scanning the running services on a target machine. The first attack was an SSH login brute-force attack and the second attack is a MySQL login brute-force attack. Each attack requires certain settings to be made, which include setting the target host IP address, usernames, and passwords that will be used for attempting to break into the target's machine and attack-specific services. SSH service uses well-known port 22, while the default port for MySQL service is 3306. The second attack was conducted a few minutes after the first one finished.

Security Onion collected network traffic information and stored it in the Elasticsearch index. Packets were analyzed by Suricata, which was able to identify threats and raised alerts about two specific events. Rules that were triggered this time were:

- *ET SCAN Suspicious inbound to mySQL port 3306*
- *ET SCAN Multiple MySQL Login Failures Possible Brute Force Attempt*
- *ET SCAN Potential SSH Scan*
- *ET SCAN Potential SSH Scan OUTBOUND*

After the index data is parsed and processed, labels were extracted using Suricata alerts, and an input dataframe was prepared for neural network analysis. Reconstruction error was computed using a trained autoencoder and input dataframe. After comparing samples from the reconstruction error array with the threshold, the following results were obtained:

- 1807 samples were classified as normal.
- 240 samples were classified as anomalies.

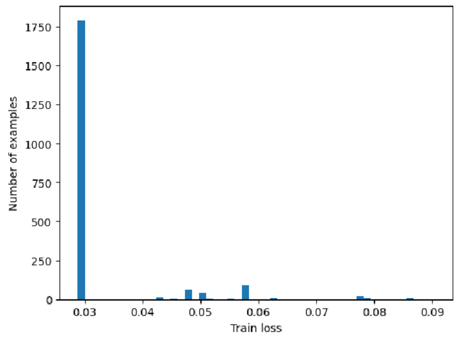

Figure 5.5: Attack launching experiment – Reconstruction error plot.

In this experiment, the True Positive Rate (TPR) and False Positive Rate (FPR) were calculated to evaluate the performance of the neural network in detecting brute-force attacks. The results of TPR and FPR were encouraging, indicating that the neural network has performed well in identifying brute-force attacks. TPR and FPR for this experiment are shown in the table below.

| **Threshold** | **TPR** | **FPR** |
|---|---|---|
| 0.04612 | 0.9864 | 0.0121 |

Table 5.3: Experiment 2 – True Positive Rate (TPR) and False Positive Rate (FPR).

## 5.4 Evaluation

In the final part of testing the implemented neural network model, the performance of the model will be tested using traditional model evaluation techniques. Model evaluation techniques are used to assess the performance of a trained model on unseen data. Some commonly used evaluation techniques include confusion matrix analysis, precision, recall, F1-score, and receiver operating characteristic (ROC) curves. These methods can guide future improvements in the form of changes to the architecture or the learning process and provide information about the strengths and limitations of the model.

### 5.4.1 Testing data

To assess neural network performance, an evaluation dataset was created. This dataset has anomalous samples obtained during the previous experiment. Data from a specific time range was collected from the Elasticsearch server and prepared for analysis. Alerts, that were raised in the same period were also collected. These alerts will be used to correlate the autoencoder prediction outputs with the anomalies that Suricata has identified using the rules.

### 5.4.2 ROC curve

A Receiver Operating Characteristic (ROC) curve is a graphical representation of a relation between True Positive Rate (TPR) and False Positive Rate (FPR). It is used to illustrate the prediction performance of a neural network using a range of values for the threshold. ROC is used as a probability curve. Another performance measurement indicator is Area Under The Curve (AUC), which represents the degree or measure of separability [18]. A high AUC indicates high performance in identifying anomalous samples and a low False Positive Rate.

An array of threshold values ranging from 0 to 1 with a step of 0.01 is created. For each threshold value in the array, TPR and FPR are calculated. Additionally, the Area Under The Curve (AUC) is calculated using lists containing TPR and FPR values for all threshold values.

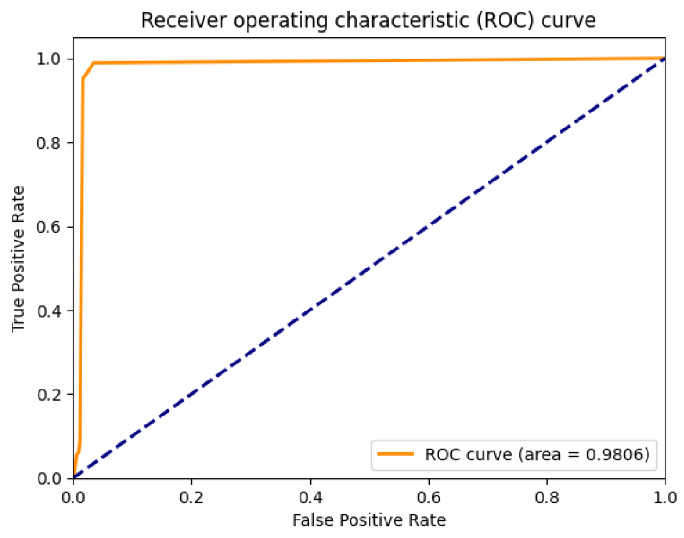

Figure 5.6: Receiver Operating Characteristic (ROC) curve.

### 5.4.3 Numerical evaluation metrics

Besides the Receiver Operating Characteristic (ROC) curve, that are other evaluation metrics that can be utilized to assess the performance of the neural network, specifically the binary classification model. Metrics, that were used for model evaluation are available in `sklearn` library. Evaluation metrics used for this purpose are [11]:

- **Accuracy**: Accuracy represents the percentage of correct predictions out of all predictions made by the neural network model.
- **Precision**: Precision is the percentage of true positive predictions out of all positive predictions made by the neural network model. It measures the model's ability to avoid false positives.

- **Recall**: Recall represents the percentage of true positive predictions out of all actual positive samples in the data. It measures the model's ability to identify all positive samples.

- **F1 score**: The F1 score is the harmonic mean of the precision and recall.

F1 score was used to calculate the optimized value for the threshold (see 5.7). In order to determine the most acceptable threshold value, the F1 score is calculated against the array of threshold values in the range $[0, 1]$ with a step of 0.01 for each value.

| **Metric** | **Value** |
|---|---|
| Accuracy | 0.9832 |
| Precision | 0.4005 |
| Recall | 0.95 |
| F1 score | 0.5634 |

Table 5.4: Evaluation metrics – calculated values.

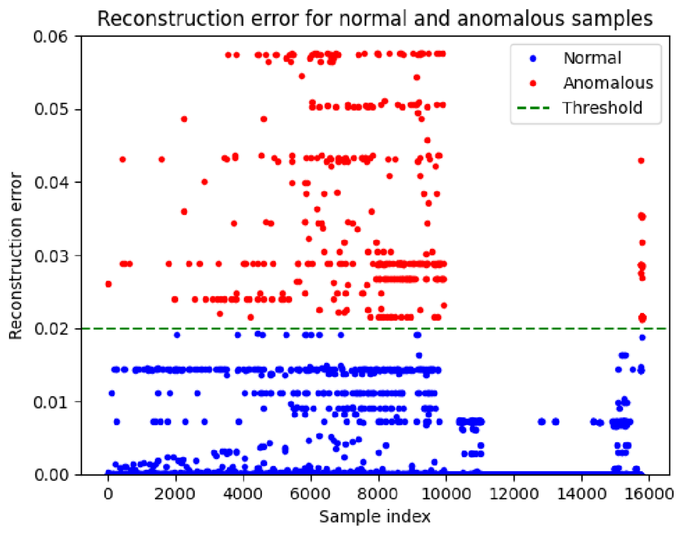

Figure 5.7: Evaluation – Classified prediction outputs with an optimized threshold value.

# Bibliography

[1] Ahmad, Z., Shahid Khan, A., Nisar, K., Haider, I., Hassan, R. et al. Anomaly Detection Using Deep Neural Network for IoT Architecture. *Applied Sciences*. 2021, vol. 11, no. 15. DOI: 10.3390/app11157050. ISSN 2076-3417. Available at: https://www.mdpi.com/2076-3417/11/15/7050.

[2] Ahmed, M., Naser Mahmood, A. and Hu, J. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*. 2016, vol. 60, p. 19–31. DOI: https://doi.org/10.1016/j.jnca.2015.11.016. ISSN 1084-8045. Available at: https://www.sciencedirect.com/science/article/pii/S1084804515002891.

[3] Analysis, M. T. *Malware Traffic Analysis* [online]. [cit. 2023-05-08]. Available at: https://www.malware-traffic-analysis.net/.

[4] Berg, R. G. van den. *Normal Distribution – Quick Introduction* [online]. SPSS [cit. 2023-01-29]. Available at: https://www.spss-tutorials.com/normal-distribution/.

[5] Bhattacharyya, D. K. and Kalita, J. K. *Network Anomaly Detection: A Machine Learning Perspective*. 1st ed. Chapman and Hall/CRC, 2013. ISBN 1466582081.

[6] Brandao, P. R. and Nunes, J. Extended Detection and Response Importance of Events Context. *Kriativ-tech*. 2021, vol. 1, no. 15. DOI: 10.31112/kriativ-tech-2021-10-58. ISSN 1646-9976. Available at: https://www.kriativ-tech.com/?p=66381.

[7] Center, I. T. R. *Annual number of data compromises and individuals impacted in the United States from 2005 to first half 2022 [Graph]* [online]. 2022 [cit. 2023-01-30]. Available at: https://www.statista.com/statistics/273550/data-breaches-recorded-in-the-united-states-by-number-of-breaches-and-records-exposed/.

[8] Chandola, V., Banerjee, A. and Kumar, V. Anomaly Detection: A Survey. *ACM Comput. Surv.* University of Minnesota. july 2009, vol. 41. DOI: 10.1145/1541880.1541882.

[9] Chandra, A. and Arya, S. K. *Demystifying Qbot Malware* [online]. Trellix, august 2022 [cit. 2023-05-08]. Available at: https://www.trellix.com/en-us/about/newsroom/stories/research/demystifying-qbot-malware.html.

[10] Chen, Z., Liu, J., Gu, W., Su, Y. and Lyu, M. R. *Experience Report: Deep Learning-based System Log Analysis for Anomaly Detection*. 2022.

[11] Czakon, J. *24 Evaluation Metrics for Binary Classification* [online]. neptune.ai, april 2023. Available at: https://neptune.ai/blog/evaluation-metrics-binary-classification.

[12] Duffield, N., Haffner, P., Krishnamurthy, B. and Ringberg, H. Rule-Based Anomaly Detection on IP Flows. In: *IEEE INFOCOM 2009*. 2009. DOI: 10.1109/INFCOM.2009.5061947.

[13] Firstbrook, P. and Lawson, C. *Innovation Insight for Extended Detection and Response* [online]. 2020 [cit. 2023-01-30]. Available at: https://www.gartner.com/en/documents/3982247.

[14] Gormley, C. and Tong, Z. *Elasticsearch: The Definitive Guide*. 1st ed. O'Reilly Media, Inc., 2015. ISBN 9781449358549.

[15] Hope, T., Resheff, Y. S. and Lieder, I. *Learning TensorFlow: A Guide to Building Deep Learning Systems*. 1st ed. O'Reilly Media, Inc., 2017. ISBN 9789352136100.

[16] Landauer, M., Onder, S., Skopik, F. and Wurzenberger, M. *Deep Learning for Anomaly Detection in Log Data: A Survey*. 2022.

[17] LLC, R. *Metasploit* [online]. [cit. 2023-05-08]. Available at: https://www.metasploit.com/.

[18] Narkhede, S. *Understanding AUC - ROC Curve* [online]. Towards Data Science, june 2016. Available at: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[19] Patil, S. U. *Loss Functions and Optimizers in ML models* [online]. GeekCulture, january 2023 [cit. 2023-05-08]. Available at: https://medium.com/geekculture/loss-functions-and-optimizers-in-ml-models-b125871ff0dc.

[20] Solutions, S. O. *Security Onion Documentation Release 2.3* [online]. 2023 [cit. 2023-05-08]. Available at: https://docs.securityonion.net/en/2.3/.

[21] Stallings, W. *Network Security Essentials: Applications and Standards (4th Edition)*. 4th ed. Pearson, 2010. ISBN 0-13-610805-9.

[22] Viglione, M. *Suricata: What is it and how can we use it* [online]. INFOSEC, march 2022 [cit. 2023-01-28]. Available at: https://resources.infosecinstitute.com/topic/suricata-what-is-it-and-how-can-we-use-it/.