



## **Ελληνικό Μεσογειακό Πανεπιστήμιο**

**Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών**

**Πρόγραμμα Σπουδών Μηχανικών Πληροφορικής ΤΕ**

### **Τίτλος:**

**Σχεδιασμός και υλοποίηση συστήματος υποστήριξης αποφάσεων  
για την εποπτεία και διαχείριση συμβάντων κακόβουλων επιθέσεων  
σε δικτυακές υποδομές**

**Κλάδος Στυλιανός (4017)**

**Επιβλέπων εκπαιδευτικός : Μαρκάκης Ευάγγελος**

**Επιτροπή Αξιολόγησης :**

- **Μαρκάκης Ευάγγελος**
- **Παναγιωτάκης Σπυρίδων**
- **Παπαδάκης Νικόλαος**

**Ημερομηνία παρουσίασης: 29/09/2020**

# Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τους καθηγητές μου Καθ. Ευάγγελο Πάλλη, Δρ. Ευάγγελο Μαρκάκη και τον υποψήφιο διδάκτορα κ. Ιωάννη Νικολουδάκη για την καθοδήγηση και την βοήθεια τους. Όλοι τους συνέβαλαν στην επιτυχή πορεία της πτυχιακής μου εργασίας καθώς και στην εκπόνηση της όπως επίσης και στην περαιτέρω ενασχόληση μου με τον κλάδο της έρευνας πάνω στον τομέα τού cybersecurity. Τέλος, θα ήθελα να ευχαριστήσω όλο το προσωπικό του εργαστηρίου Pasiphae Lab που με βοήθησαν οποιαδήποτε στιγμή το χρειαζόμουν.

# Abstract

The growing dependence on cyberspace, which has increased over the last decade, and the rapid advancement in automated cyber-attacks require every Information Technology expert, such as network administrators, to be able to detect every cyber-incident and mitigate it at the earliest opportunity. For this to be possible, these circumstances demand the development of frameworks that can assist with such incidents. Lately, there is a pattern for the turn of events for these kinds of frameworks, based on "Situational Awareness". Supported by concepts of this theory, our research involves collecting information from heterogeneous sensors, which provide further assistance to network administrators so that they can have a comprehensive overview of what is happening across a network.

This thesis elaborates upon the design and development of a neural network that detects malicious traffic and is trained on an enhanced dataset, which includes not only network-related data but also data obtained from a vulnerability assessment tool. This thesis aims to prove that a neural network, with multiple heterogeneous data inputs, provides more accurate prediction rates than a neural network which is solely trained on one type of data, in this case, network traffic data. The results of our experiment showcase that a neural network that was trained with multiple heterogeneous data inputs, has an increase in accuracy by almost 2%, in comparison with a neural network which is trained with normal network traffic data.

# Περίληψη

Η συνεχώς αυξανόμενη εξάρτηση των ανθρώπων από τον κυβερνοχώρο, η οποία έχει αυξηθεί κατά πολύ την τελευταία δεκαετία, καθώς και η ραγδαία εξέλιξη των αυτοματοποιημένων κυβερνο-επιθέσεων απαιτούν από κάθε επαγγελματία του τομέα της Πληροφορικής, όπως διαχειριστές δικτύων, να είναι σε θέση να ανιχνεύει και να επιλύει οποιοδήποτε ζήτημα κυβερνο-επίθεσης όσο το δυνατόν γρηγορότερα. Προκειμένου να είναι εφικτό κάτι τέτοιο, υπάρχει η ανάγκη για την ανάπτυξη συστημάτων τα οποία παρέχουν βοήθεια σε αυτήν την διαδικασία. Τα τελευταία χρόνια, υπάρχει μία τάση, τέτοια συστήματα να αναπτύσσονται βασιζόμενα σε μία θεωρία η οποία ορίζεται ως «Επίγνωση της Κατάστασης» (Situational Awareness). Βασιζόμενοι πάνω σε στις έννοιες αυτής της θεωρίας, η έρευνά μας περιλαμβάνει τη συλλογή δεδομένων από ετερογενείς αισθητήρες, ώστε να παρέχει περαιτέρω υποστήριξη στους διαχειριστές δικτύου, με αποτέλεσμα να έχουν μία συνολική εικόνα του τι συμβαίνει στο δίκτυο.

Σε αυτήν την πτυχιακή, σχετίζεται με τον σχεδιασμό και την δημιουργία ενός νευρωνικού δικτύου το οποίο ανιχνεύει κακόβουλη δικτυακή κίνηση και έχει εκπαιδευτεί με τη χρήση ενός βελτιωμένου σετ δεδομένων (dataset), το οποίο, πέρα από δικτυακά δεδομένα, περιέχει δεδομένα και από την αξιολόγηση ευπαθειών (vulnerability assessment). Σκοπός αυτής της πτυχιακής εργασίας είναι να αποδείξει πως ένα νευρωνικό δίκτυο, εκπαιδευμένο από ετερογενή δεδομένα, παρέχει πιο αξιόπιστες προβλέψεις από ένα νευρωνικό δίκτυο το οποίο έχει εκπαιδευτεί αποκλειστικά με τη χρήση δικτυακών δεδομένων. Τα αποτελέσματα της έρευνάς μας, αποδεικνύουν πως ένα νευρωνικό δίκτυο, το οποίο έχει εκπαιδευτεί με ετερογενή δεδομένα, αποδίδει σχεδόν 2% καλύτερα, σε σύγκριση με ένα νευρωνικό δίκτυο που έχει εκπαιδευτεί μόνο με δικτυακά δεδομένα.

## Table of Contents

Abstract.....	3
Περίληψη.....	4
Table of Contents.....	5
1. Introduction.....	8
2. State of the Art.....	10
3. Technology Enablers.....	13
3.1 Vulnerability Assessment.....	13
3.2 Machine Learning.....	14
3.2.1 TensorFlow 2.0.....	14
3.2.2 Keras.....	15
3.2.3 Weka.....	16
3.3 Development languages and frameworks.....	17
3.3.1 Python3.....	17
3.4 Cybersecurity Tools, Frameworks and Operating Systems.....	18
3.4.1 CICFlowMeter 4.0.....	18
3.4.2 Metasploit Framework.....	20
3.4.3 Metasploitable 3.....	21
3.4.4 Kali Linux 2020.2.....	21
3.4.5 Ubuntu Vulnerable Server.....	21
3.4.6 Wireshark / TCPDump.....	21
4. Implementation.....	23
4.1 Environment Setup.....	23
4.1.1 Windows Server 2008 R2.....	23
4.1.2 Ubuntu Server 20.04.....	24
4.1.3 Kali Linux 2020.2.....	24
4.2 Concept.....	25
4.2.1 Architectural Elements.....	26
4.3 Dataset Creation.....	27
4.4 Machine Learning Model Development.....	30
4.5 Use Case.....	37

5. Evaluation.....	38
5.1 Aim.....	38
5.2 Method.....	39
5.3 Variables.....	39
5.3.1 Dependent Variables.....	39
5.3.2 Independent Variables.....	39
5.3.3 Fixed Variables.....	39
5.4 Prediction.....	40
5.5 Results.....	40
5.6 Discussion.....	41
5.7 Evaluation.....	42
6. Conclusion.....	42
7. References.....	44

## List of Figures

Figure 1 : Some of the available OpenVAS reports.....	16
Figure 2 : Types of machine learning.....	17
Figure 3 : Simple representation of a Multilayer Perceptron network topology.....	18
Figure 4 : Weka's graphical interface.....	19
Figure 5 : CICFlowMeter's graphical interface.....	21
Figure 6 : Code snippet from Metasploit Framework's Python3 API.....	22
Figure 7 : Wireshark graphical interface.....	24
Figure 8 : Representation of the network topology.....	25
Figure 9 : Conceptual Diagram.....	27
Figure 10 : 4th Iteration's Model Accuracy.....	33
Figure 11 : 9th Iteration's Model Accuracy.....	35
Figure 12 : 9th Iteration's Model Loss.....	35
Figure 13 : Model #2 Accuracy.....	38
Figure 14 : Model #2 Loss.....	38
Figure 15 : Situational Awareness Framework Sequence Diagram.....	40
Figure 16 : Graph representation of Models' Predictions on live attacks.....	42

## **List of Tables**

Table 1.....	30
Table 2 : Parameters changed throughout the iterations, and Accuracy and Loss produced.....	35
Table 3 : Comparison of accuracy and loss between the two different trained models.....	36
Table 4 : Vulnerabilities and their corresponding CVE entries and attack types.....	40
Table 5 : Model's Predictions on live attacks.....	41



# 1. Introduction

The lack of knowledge on cybersecurity is a common phenomenon for regular users of an enterprise and corporate networks that can lead to cybersecurity incidents that can lead to serious financial losses and data theft.

Also, more and more organizations and businesses store, process and exchange sensitive information, which is very valuable, to malicious actors [1]. During the recent years, cyber-attacks have been quite frequent and pose a grave threat to enterprises of different sizes, forcing several of them to bankruptcy, and eventually closure. For this reason, network administrators are required to have full awareness of their network's activity, as well as a strong knowledge of proactive cybersecurity protection techniques and procedures, and cybersecurity incident mitigation strategies [2] [3] [4] [5]. Until this day, traditional intrusion detection and prevention systems are utilized, but their limitations impair their effectiveness against several malicious activities. One major limitation is that most of these systems utilize only a certain type of data as an input, to generate their results. Moreover, the intricacy and automation of cyber-attacks are increasing daily.

According to Endsley et.al, Situational Awareness is the ability to apprehend the environmental components and incidents relating to time or space, to realize their significance, and to prognosticate their forthcoming condition [6]. This term by default refers to military applications but in 1999, Bass Tim first stated that the future of Cybersecurity would be in the application of Situational Awareness theorem [7], [8]. He recommended that the next generation of intrusion detection systems will fuse data from heterogeneous sensors for the creation of cyberspace situational awareness.

In consideration of the foregoing, Situational Awareness in Cybersecurity allows network administrators or security analysts to take advantage of heterogeneous data from different sources, such as network traffic data and vulnerability assessments, to gain an understanding of the surrounding environment. On top of that, the interpretation of that information provides insight and knowledge of the network, while assisting to predict what might happen in the foreseeable future.

Situational Awareness in Cyber Security is addressed in various ways in the literature. Several investigations purpose methods to fuse data from heterogeneous sources, while others purpose machine learning techniques to automate the assessment process. Furthermore, multiple surveys present mathematical and probabilistic approaches.

Even though all of those studies propose novel utilizations of Situational Awareness in Cyber Security, most implementations depend on human intervention. Moreover, heterogeneous data fusion approaches do not utilize intelligence-based techniques.

Therefore, to address the aforementioned issues, this thesis proposes an ML-powered Situational awareness (SA) framework that implements heterogeneous data fusion, provides a dynamic way of operation, and as a result, it demands little, to no human intervention. The awareness is achieved by utilizing a neural network, which has been trained on an enhanced network dataset, with both network data and the system's vulnerabilities. The results of our experiments propose that this neural network is more efficient, by almost 2%, regarding predictions than a neural network, trained with a simple dataset, with only network data.

The rest of the thesis is structured as follows. Section 2, presents the current state of the art concerning Situational Awareness in the Cyber Security domain. Section 3, introduces the technology enablers and explains the reasons why we selected them. In Section 4, we present our ML-powered Situational Awareness framework. Section 5, presents the evaluation process that was followed, to assess our framework. Finally, Section 6, summarizes this thesis, reviews the results of our experiment and presents the suggested future work.

## 2. State of the Art

Various research initiatives have been conducted in order to realize how Situational Awareness can be handled, within the Cyber Security field. In this section, we will present a brief review of the most pertinent and recent literature on that subject.

Tero Kokkonen et al. [9] proposed a novel Architecture for a Cyber Security Situational Awareness System, based on multi-sensor data-fusion components and data sharing with trusted associates. Their architecture utilizes every level of the data fusion model, proposed by the US Joint Directors of Laboratories (JDL) [10]. The architecture consists of a Data Fusion engine, a Human-Machine interface, and a Visualization layer. A security analyst is responsible for the configuration of the sensors and for analyzing the sensor feed.

Yanfa Zhang et al. [11] presented a visual analysis technology for Cyber Security Situational Awareness, which can display features and visual analysis of the network topology providing dynamic query and human-computer interaction. Their proposed Network Visualization Analysis tool is refined in comparison with their previous research [12] [13].

H. Park et al. [14] suggested an alternative course of action for Cyber Situational Awareness, combining Regular Expressions and a proposed Evaluation Methodology. This method overcomes the common problem that occurs when a system uses signature-based pattern matching, which is the limited string-based matching that most pattern matching algorithms use. The collection of the regular expressions that create the detection rules is coordinated by a security analyst.

Elena Dynikova et al. [15] proposed a Common Vulnerability Scoring System (CVSS)-based Probabilistic Risk Assessment for Cyber Security Situational Awareness and Countermeasure Selection. The assessment starts by calculating attack probabilities and risks, by using their proposed technique. Following, the system generates input data by simulating attack sequences, followed by the recalculation of the attack probabilities and risks after every simulated attack. Then the assessment continues by comparing the calculated risk levels with compromised attack graph nodes. Finally, the comparison between the forecasted attack steps with real attack steps occurs. This is a solely probabilistic/mathematical approach.

Tomas Jirsik et al. [16] proposed a Cyber Situational Awareness approach that includes IP Flow monitoring and is based upon the improvement of the three primary issues related to this matter. In light of the issue of host recognition in unencrypted traffic, their strategy includes the

development of methods with regard to host identification. Regarding the lack of network visibility and comprehension, they suggest and assess IP Flow monitoring methods that heighten network perception. Furthermore, in their method they offer an alternative for decreasing IP Flow monitoring delays. In terms of overcoming the challenges of raw data overloading, context deficiency, low speed of response, and the unified view of a network, Tomas Jirsik et al. [17] proposed a novel Network Cyber Situational Awareness framework. This framework leverages a distributed data-stream processing system and handles big-data instantly, by utilizing a stream-based approach instead of a batch-based one, resulting in the diminution of the required time for response. However, this method resulted in that the network data that has already been processed, cannot be analyzed further after the stream is finished.

X. Liu et al. [18] [19] presented a prototype of a Multiclass Support Vector Machine (SVM)-based fusion engine. Their conclusive outcomes suggest that the SVMs have potential in real-time Intrusion Detection System (IDS) applications that enhance network security situational awareness, due to the faster real-time results and similar reliability of the SVM [20] approach to the Multi-Layer Feed Forward Neural Network (MLF-NN).

Wang et al. [21] provided a thorough description of how they utilized their neural networks. During the data aggregation, they distinguished that some features needed to be excluded in order to prevent unnecessary data fields, of each of the IDSs, from weighing down the neural network. They suggested the use of a MLF-NN [22] and their applied algorithm was trained and evaluated by using the DARPA 1999 data set. A rapid fusion and assessment of the intrusion behavior, identified in the traffic, is being produced by the output.

M. L. Mathews [23] et al. presented a cooperative procedure to Situational Awareness for Cyber Security. Their system architecture is composed of a collaborative situation-aware IDS that collects data from individual sensors. This is based on earlier work from the same group [24] [25] that consists of an ontology. Weka was utilized for the learning and classification process.

Huan Wang et al. [26] proposed a network security situational assessment model that is based upon an order of importance and combines the Analytic Hierarchy Process with the D-S Evidence Theory [27] for the fusion of multi-source equipment. They simplify the situational assessment problem while using the DARPA99 [28] dataset.

Yuangang Yao et al. [29] proposed a data fusion framework based on an attack model. The attack model that has been used in this study, is able of altering the attacks that are happening across the network into organized and normalized knowledge, which is used for representation subsequently. Connection between network attacks is described by first-order logical assumptions and numerical models are being used regarding the analysis of security incidents. The attacker's single-time behavior is being specified by the framework, which utilizes numerous low-level alarm information in order to be able to achieve identification. Moreover, this framework is capable of matching distinct attack actions and extracting the objective of the attack by using casual chain reasoning. Their framework's output includes an abstract attack pattern. Interesting extensions of their work contain studying the natural language and knowledge maps so that the accuracy of matching attack patterns will be increased.

As our literature review indicated, Situational Awareness in Cybersecurity is approached differently by different research initiatives. Some case studies suggest approaches that include the use of machine learning algorithms, but they utilize outdated and possibly obsolete training datasets. Another technique to achieve Situational Awareness, which is recommended in publications, is the fuse of heterogeneous data. However, machine learning is not utilized on any of those studies as an approach on that matter. Furthermore, most of the aforementioned proposals require human intervention to a great extent.

To address those issues, we propose a Cyber Security Situational Awareness framework that:

- provides a fully automated Cyber Security Situational Awareness lifecycle, by utilizing automated scripts
- functions with a data fusion logic, by combining different data from heterogeneous sources to gain the required environmental awareness
- utilizes Machine Learning techniques as a means to assess the gathered information and produce a prediction on possible cyber security incidents
- leverages a custom dataset, which did not undergo any modifications, in order to represent the network traffic during the attacks, as realistically as possible

## 3. Technology Enablers

For the design and development of the proposed framework, several different technologies, tools and frameworks were utilized. This chapter presents all those technologies in detail. All of the technologies are Open-Source and available for free.

### 3.1 Vulnerability Assessment

Vulnerability assessment<sup>1</sup> alludes to the way toward distinguishing risks and vulnerabilities in computer networks, applications, systems, hardware and further sections of the IT environment. Through Vulnerability Assessment it can be evaluated whether the system is at risk regarding any known vulnerabilities. In addition to this evaluation, each vulnerability that is identified through this process, is assigned to a severity level. If and whenever it is possible, it recommends remediation or mitigation actions. Several types of vulnerability assessment exist, such as host, network, wireless and database assessment. In this thesis, the Open Vulnerability Assessment Scanner (OpenVAS)<sup>2</sup> was used, due to its open-source nature and overall popularity [30] [1] . OpenVAS is developed and maintained by Greenbone Networks and is a comprehensive vulnerability assessment system that can detect security issues in various types of terminals and services. It falls into the category of Network assessment. It utilizes a large database of known vulnerabilities and compiles an assessment report in a variety of formats. Some of the report's formats are illustrated in Figure 1. For this thesis, we built a RESTful API on top of the OpenVas

---

<sup>1</sup> [https://en.wikipedia.org/wiki/Vulnerability\\_assessment](https://en.wikipedia.org/wiki/Vulnerability_assessment)

<sup>2</sup> <https://www.openvas.org/>

platform, to manage it programmatically. This way our automated scripts, perform vulnerability assessments and retrieve the results in machine-readable format (JSON).

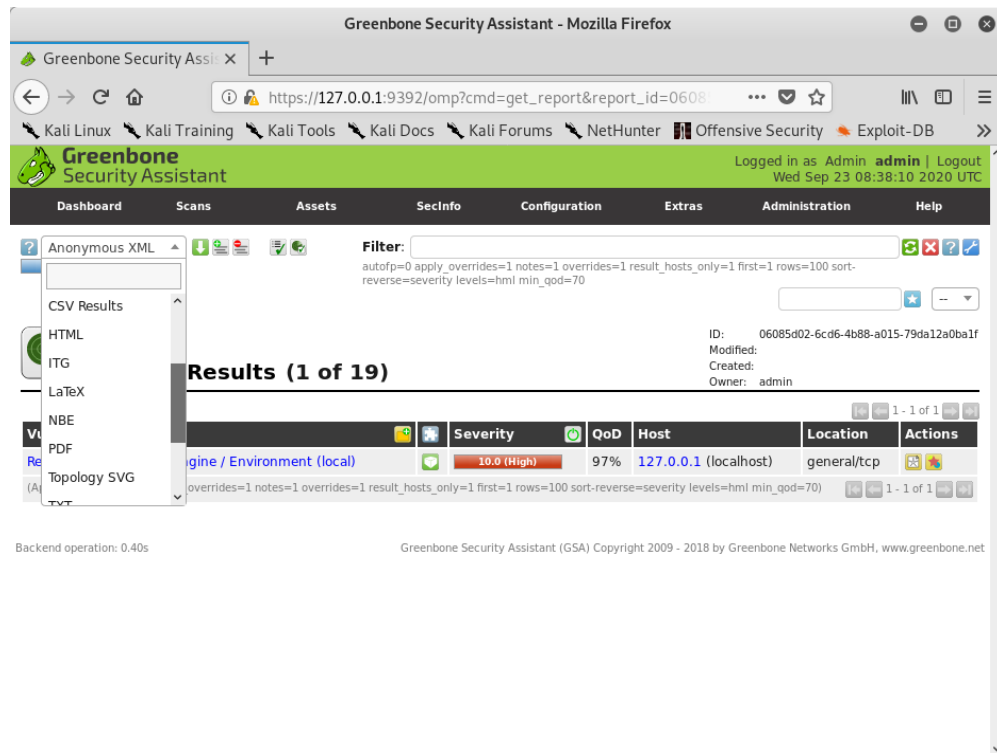


Figure 1 : Some of the available OpenVAS reports

### 3.2 Machine Learning

Our proposal for a framework that operates without human intervention could not be possible without the use of a Machine Learning model. Machine Learning<sup>3</sup> is a field of study that contains data analytics techniques and computational methods that enable computer systems to acquire knowledge and improve from experience like humans, without the requirement of explicit programming instructions. Through the use of machine learning algorithms, computer systems are being trained to automatically transform the acquired knowledge into usable models, with minimal or even without human interference or support. There are many Machine Learning Types [31] (Figure 2). The three most popular ones that are used in Learning Problems are: i) Supervised Learning, ii) Unsupervised Learning, and iii) Reinforcement Learning. Which type will be used, depends on factors like the size, quality, nature of data, the available computational time, and what the desired results are. Supervised Learning is mostly used for regression, forecasting, or regression. Unsupervised Learning is used for Clustering or Dimension Reduction. Reinforcement Learning is set to work without a dataset, and it optimizes itself based on the behavior of an agent inside an environment. For this research, Supervised Learning was chosen, because our research subject lies in the group of Multi Classification problems, and for the reason that the data processed is labelled beforehand.

<sup>3</sup> [https://en.wikipedia.org/wiki/Machine\\_learning](https://en.wikipedia.org/wiki/Machine_learning)

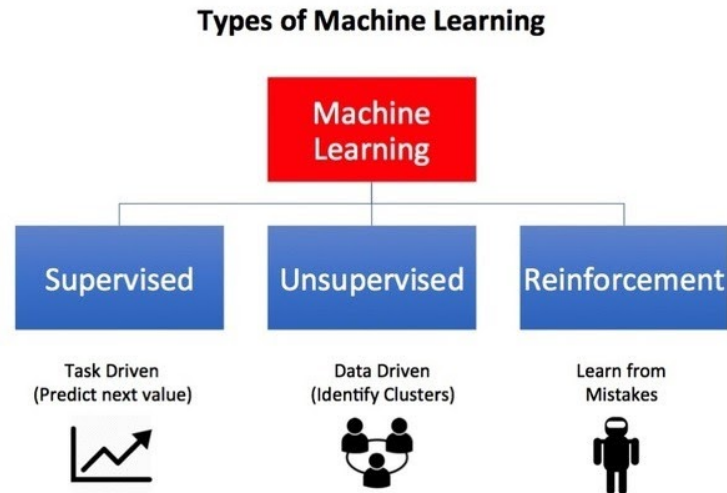


Figure 2 : Types of machine learning

### 3.2.1 TensorFlow 2.0

While coping with the development of a Machine Learning model, we came across the necessity of a Machine Learning library. For this purpose, we chose to use TensorFlow 2.0<sup>4</sup>, which is an Open Source library, developed by Google, for numerical computations and large-scale Machine Learning, with a strong developer community support. TensorFlow's library is composed of both Machine Learning and Deep Learning algorithms and models. It has a dataflow architecture that offers users the ability of generating dataflow graphs. These data structures may represent a series of processing nodes or a computational graph and they depict the motion of data through the structure. Furthermore, the nodes in these data structures correspond to mathematical operations and multidimensional arrays serve as connections among these nodes. These multidimensional arrays are otherwise called tensors<sup>5</sup>. TensorFlow 2.0 was selected over other options like PyTorch<sup>6</sup>, Theano<sup>7</sup> and scikit-learn<sup>8</sup>, because of its advantage of allowing for the development of Machine Learning models, in abstraction. Moreover, this library includes the TensorFlow Serving<sup>9</sup>, which is a service that enables any trained and saved model to be used through a REST API, with external data via POST HTTP Requests. Finally, TensorFlow 2.0 is already integrated with Keras. TensorFlow 2.0 and Keras are exclusively used for the creation of the Neural Network algorithm used in this research.

<sup>4</sup> <https://www.tensorflow.org/>

<sup>5</sup> <https://en.wikipedia.org/wiki/Tensor>

<sup>6</sup> <https://pytorch.org/>

<sup>7</sup> <http://deeplearning.net/software/theano/>

<sup>8</sup> <https://scikit-learn.org/stable/>

<sup>9</sup> <https://www.tensorflow.org/tfx/guide/serving>



### 3.2.2 Keras

Keras<sup>10</sup> is an open-source Python library. One of the advantages of Keras is that TensorFlow 2.0 library is, as a matter of course, incorporated with it. Moreover, Keras library is modular and easy to use, as well as it is simple to extend, due to its uncomplexity, and to operate by using the Python programming language. Keras also contains various independent modules that developers can consolidate to create new models. Such modules include cost and activation functions, optimizers, regularization schemes and neural networks. In this thesis, we used Keras' characteristic modularity to create a Multi-Layer Perceptron Artificial Neural Network [32]. Multi-Layer Perceptrons, or most commonly named, Neural Networks (NN), are statistical models inspired by biological neural networks. They acquire the capability of learning the depiction of the observed data and identifying the optimal solution, regarding the output variable that calls for prediction. The basic components of a Neural Network are the artificial neurons. Artificial neurons are computational units and they are arranged in networks of neurons, which are called layers. Every layer represents a column of neurons, it may consist of one or multiple neurons and one network may have multiple layers. This kind of network is usually referred to as a Network Topology (Figure 3). Each artificial neuron has the ability of receiving input signals that are assigned to a specific weight. An output signal is then produced by the application of an activation function on the input signal. As soon as the Neural Network is arranged, a dataset needs to be used for the training of the network. However, before that, the data needs to be pre-processed to have the appropriate format that is needed for the training of the Neural Network. Once that process is finished, the model is ready to make a prediction.

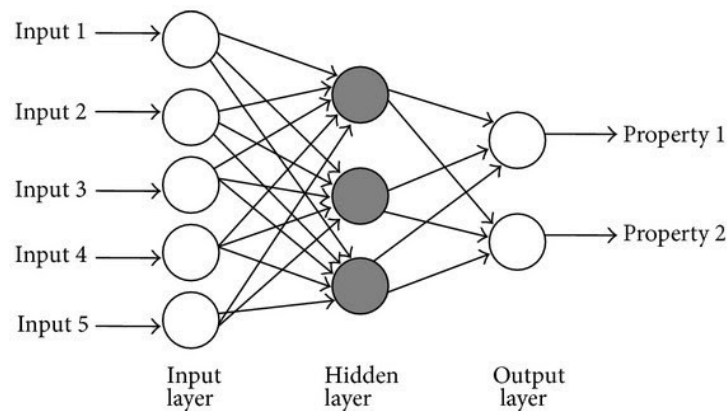


Figure 3 : Simple representation of a Multilayer Perceptron network topology

<sup>10</sup> <https://keras.io/>

### 3.2.3 Weka

For the creation of the dataset used for the training of our NN, utilization of a tool was required, to easily preview the dataset and process it further. Thus, we used Weka<sup>11</sup>, which is an open-source machine learning software with a graphical user interface that enables users to quickly assess a dataset by displaying statistics about it. Weka's role was significant in the creation of our training dataset, due to its Attribute Evaluator. Weka's Attribute Evaluator was used in the step of dimensionality reduction. One of the most popular feature selection techniques is Info Gain Attribute Evaluator, which is also the most efficient attribute evaluator for supervised learning occasions, as mentioned in [33].

Figure 4 illustrates Weka's graphical interface.

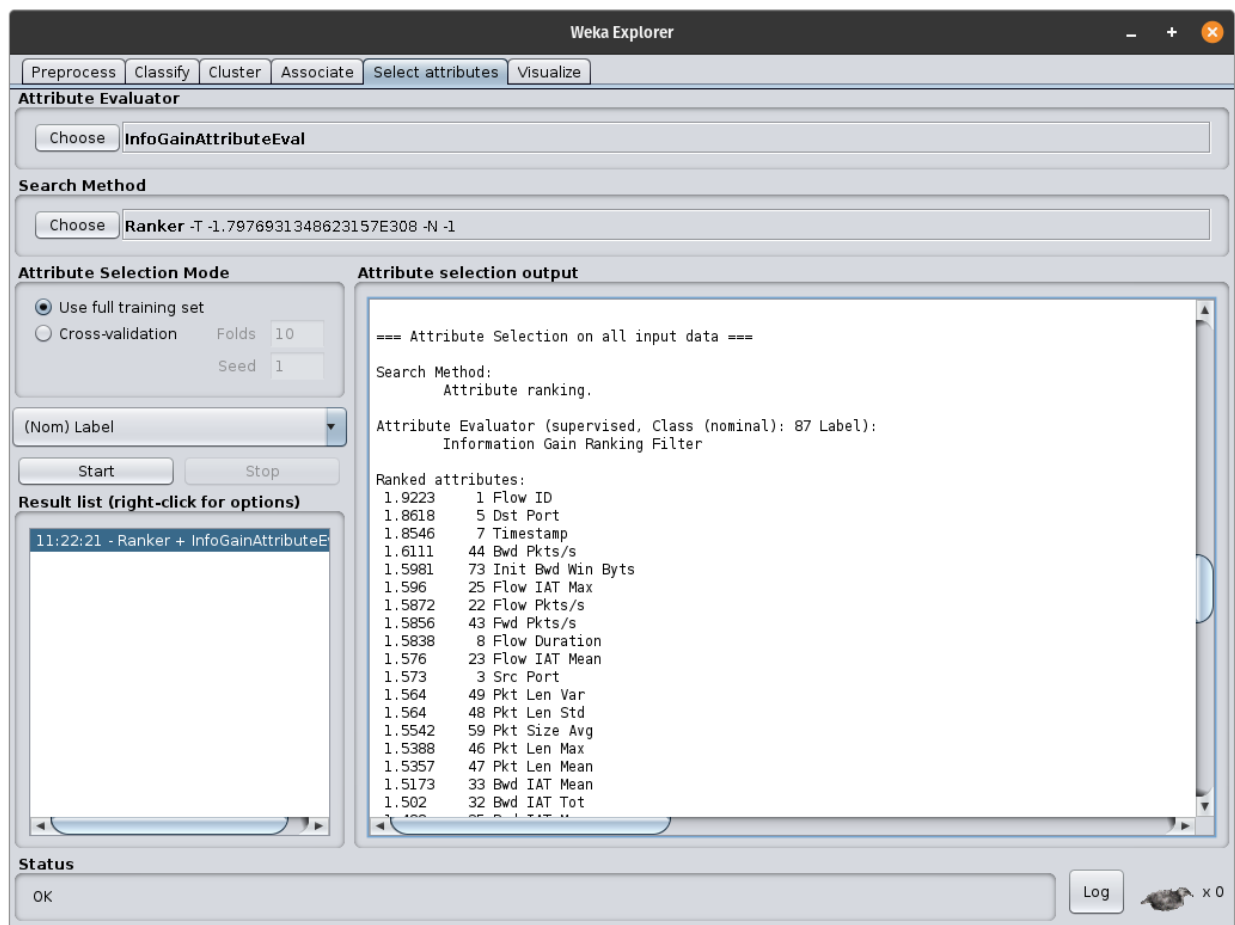


Figure 4: Weka's graphical interface

<sup>11</sup> <https://www.cs.waikato.ac.nz/ml/weka/>

### 3.3 Development languages and frameworks

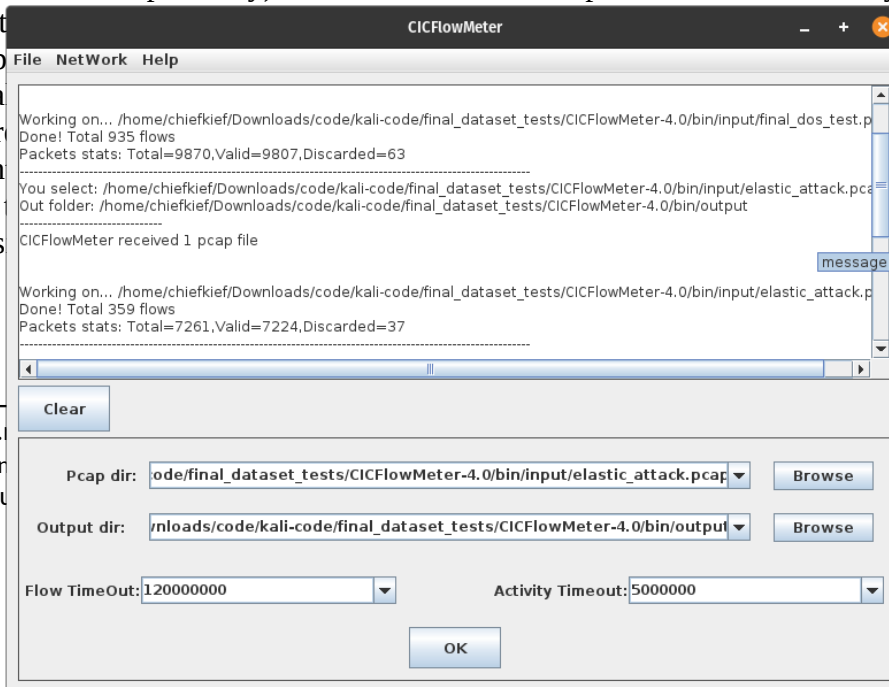
#### 3.3.1 Python3

Data preprocessing is one of the most important procedures when building a Neural Network. Data needs to be converted or encoded in order for the Machine Learning algorithm to be able to acquire knowledge from it and that is the sole purpose of data preprocessing. [34]. There are several features, of different types, that are used to describe data objects. Features can be Categorical or Numerical. Some of the data preprocessing steps are Data Quality Assessment, Feature Sampling, and Feature Encoding. For that matter, several Python3 Libraries were utilized. Pandas was the first one. Pandas [35] is an Open Source data analysis and manipulation tool. It can process Comma Separated Values (CSV) files which is the most common dataset file type. In our case, it was used to import Comma Separated Values files into the Data Fusion script from which those values were checked for missing values, inconsistent values, feature aggregation, dimensionality reduction, and feature encoding. Numpy was also used. Numpy [36] is used for scientific computing in Python. TensorFlow Serving's POST Request needs Numpy's arrays as an input to properly function, so for the prediction to be made, we have to transform the Comma Separated Values files, that we previously imported with Pandas, to Numpy Arrays and then send them to the TensorFlow Serving's API. We also used Scapy<sup>12</sup>, which is a powerful interactive packet manipulation python3 library. For this research, it is used for live packet capture. Scapy captures network traffic as a Packet Capture (pcap) type of files. Captured pcap files are then transformed into network flow files and then fed to the Tensorflow Serving API for the prediction based on the trained model.

### 3.4 Cybersecurity Tools, Frameworks and Operating Systems

#### 3.4.1 CICFlowMeter 4.0

To convert raw network traffic packet data to network flows, we used CICFlowMeter. CICFlowMeter<sup>13</sup> [37] [38] is an open-source and free network traffic flow (NetFlow) converter and analyzer. It is developed and maintained by the Canadian Institute of Cybersecurity<sup>14</sup>. CICFlowMeter generates bidirectional flows. These flows consist of packets and the first of these is used for defining both the forward and backward direction (source-to-destination and destination-to-source respectively). On the other hand, the packets are described by more than 80 statistical network flow attributes. The output of CICFlowMeter is a Comma Separated Values (CSV) file. This file provides a more efficient and faster and easier way to process network traffic data. The output of CICFlowMeter is a Comma Separated Values (CSV) file. This file provides a more efficient and faster and easier way to process network traffic data. The output of CICFlowMeter is a Comma Separated Values (CSV) file. This file provides a more efficient and faster and easier way to process network traffic data.



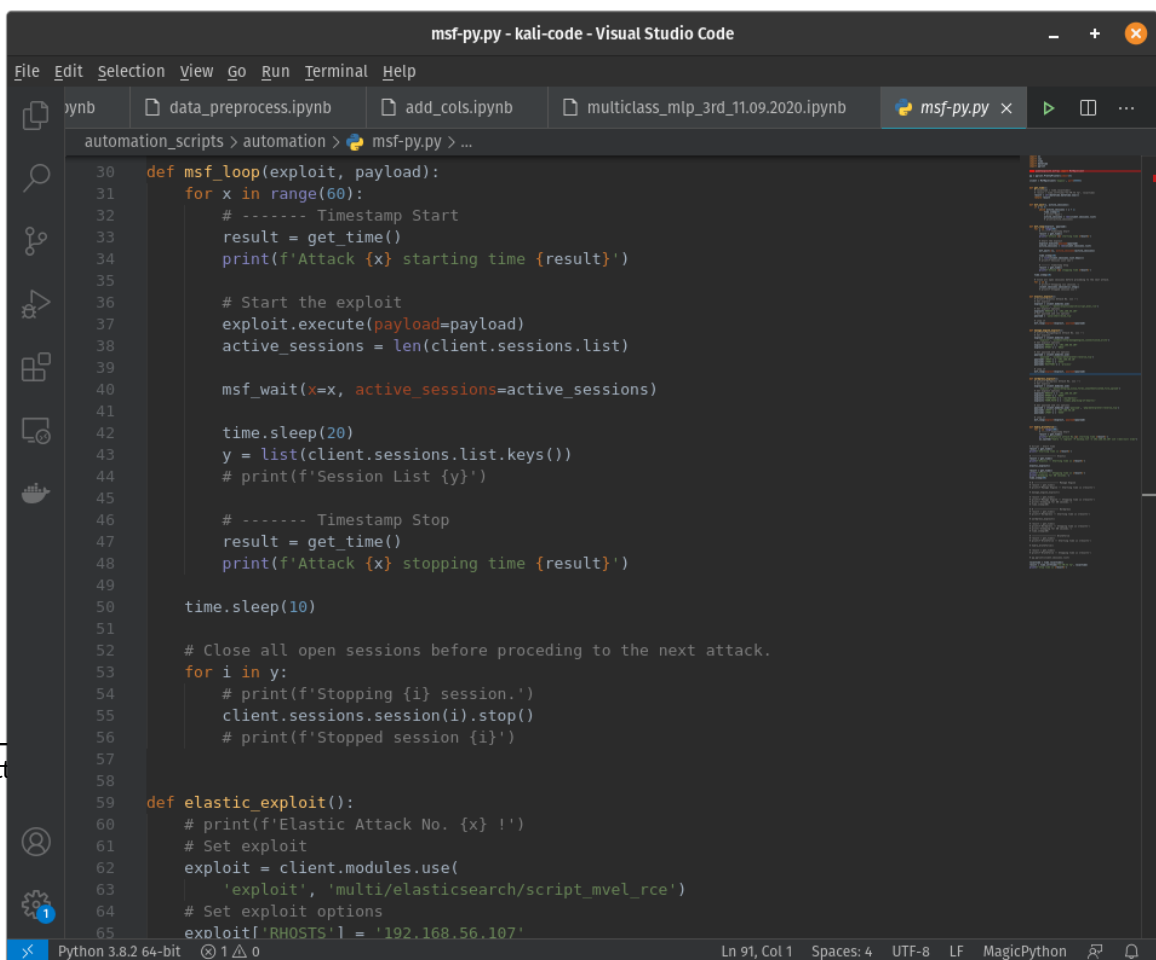
<sup>12</sup> <https://scapy.net/>

<sup>13</sup> <https://www.cicflowmeter.com/>

<sup>14</sup> <https://www.cicflowmeter.com/>

### 3.4.2 Metasploit Framework

For the simulation of realistic attacks, the automation of the attacks was mandatory to produce a dataset that would be large enough, so that the neural network would be trained properly. Metasploit Framework<sup>15</sup> (MSF) was chosen because of the Python3 API that it already has



```
msf-py.py - kali-code - Visual Studio Code
File Edit Selection View Go Run Terminal Help
automation_scripts > automation > msf-py.py > ...
30 def msf_loop(exploit, payload):
31     for x in range(60):
32         # ----- Timestamp Start
33         result = get_time()
34         print(f'Attack {x} starting time {result}')
35
36         # Start the exploit
37         exploit.execute(payload=payload)
38         active_sessions = len(client.sessions.list)
39
40         msf_wait(x=x, active_sessions=active_sessions)
41
42         time.sleep(20)
43         y = list(client.sessions.list.keys())
44         # print(f'Session List {y}')
45
46         # ----- Timestamp Stop
47         result = get_time()
48         print(f'Attack {x} stopping time {result}')
49
50         time.sleep(10)
51
52         # Close all open sessions before proceeding to the next attack.
53         for i in y:
54             # print(f'Stopping {i} session.')
55             client.sessions.session(i).stop()
56             # print(f'Stopped session {i}')
57
58
59 def elastic_exploit():
60     # print(f'Elastic Attack No. {x} !')
61     # Set exploit
62     exploit = client.modules.use(
63         'exploit', 'multi/elasticsearch/script_mvel_rce')
64     # Set exploit options
65     exploit['RHOSTS'] = '192.168.56.107'
```

<sup>15</sup> <https://www.metasploit.com/>

available (Figure 6). MSF has a large collection of exploitation scripts that are used to take advantage of well-known vulnerabilities in order to penetrate a remote machine through remote file execution. Moreover, MSF is very customizable. The purpose of using MSF in this research was, to automate certain Remote File Inclusion type attacks, which would assist us in the formation of the network dataset. We were able to extract Packet Capture (pcap) files from within the victim's virtual machine, while it was being attacked. The Python3 API was explicitly used for the automation of the attacks.

### 3.4.3 Metasploitable 3

To simulate a realistic attack scenario, a vulnerable victim machine was required. For that purpose, Metasploitable 3 was chosen. Metasploitable 3<sup>16</sup> is a pre-built virtual machine that has a large number of security vulnerabilities. Our aim here was to use Metasploitable 3 as a potential victim so that a realistic attack scenario would be emulated. When building the Metasploitable 3 vulnerable virtual machine, there is a choice between a vulnerable Ubuntu 14.04 Server and a vulnerable Windows Server 2008 R2. For this thesis Windows Server 2008, R2 was used. Metasploitable 3 was chosen because of the variety of vulnerabilities that it already has by default, and because it is very customizable, open-source, and free. These vulnerabilities are mapped with the list of Common Vulnerability Exposures (CVE)<sup>17</sup>. CVE is an inventory of entries, where each entry uniquely describes a publicly known vulnerability. Those CVE entries are also included in the U.S. National Vulnerability Database (NVD)<sup>18</sup>.

### 3.4.4 Kali Linux 2020.2

Kali Linux<sup>19</sup> is a Linux distribution, which is used by ethical cyber-security penetration testers. It contains several tools, including Metasploit Framework. Automated attacks were coordinated from the Kali Linux virtual machine with the Metasploit Framework, Hydra, which is a tool for SSH Bruteforce attack<sup>20</sup>, and a custom script to emulate a Denial of Service (DoS) Slow Loris attack [39].

### 3.4.5 Ubuntu Vulnerable Server

For capturing DoS attack network traffic, an Ubuntu 20.04 Server<sup>21</sup> was used where an HTTP web server was hosted, Apache2<sup>22</sup>. Apache2 was selected because it is used on almost 40% of the available web servers worldwide and for that reason, it fitted very well for the realistic creation of the dataset since it represents something so common.

---

<sup>16</sup> <https://github.com/rapid7/metasploitable3>

<sup>17</sup> <https://cve.mitre.org/>

<sup>18</sup> <https://nvd.nist.gov/>

<sup>19</sup> <https://www.kali.org/>

<sup>20</sup> [https://en.wikipedia.org/wiki/Brute-force\\_attack](https://en.wikipedia.org/wiki/Brute-force_attack)

<sup>21</sup> <https://releases.ubuntu.com/20.04/>

<sup>22</sup> <https://httpd.apache.org/>

### 3.4.6 Wireshark / TCPDump

For the network packet capture, we used Wireshark<sup>23</sup>, which is a free and open-source protocol analyzer, with a graphical user interface Figure 7. It allows users to perform deep inspection, for several protocols, live, and offline. Wireshark was used because of its command-line interface utility, TCPDump. TCPDump was used for the automation of the network packet capture after the model was trained. Wireshark was used for the live packet capture from Windows Server 2008 R2 virtual machine, while its command-line interface, TCPDump, was used for the live packet capture from the Ubuntu 20.04 Server.

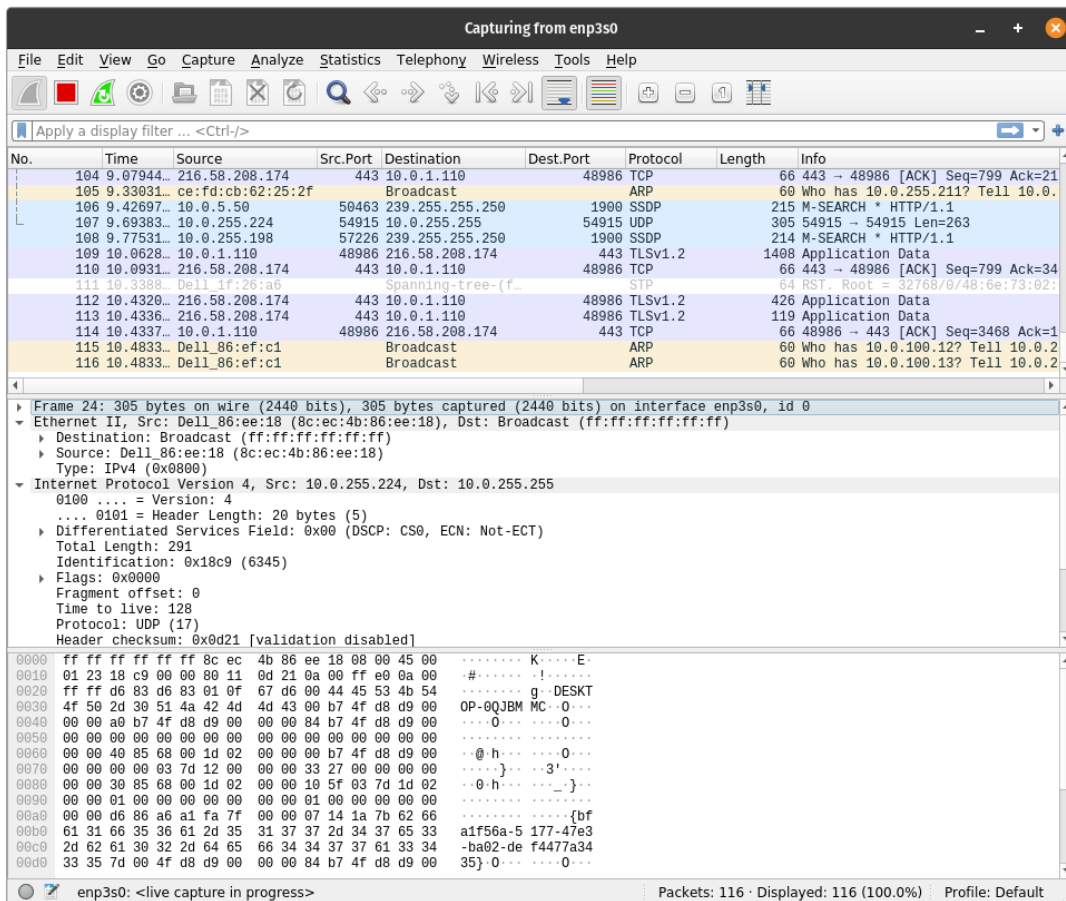


Figure 7 : Wireshark graphical interface

<sup>23</sup> <https://www.wireshark.org/>



## 4. Implementation

### 4.1 Environment Setup

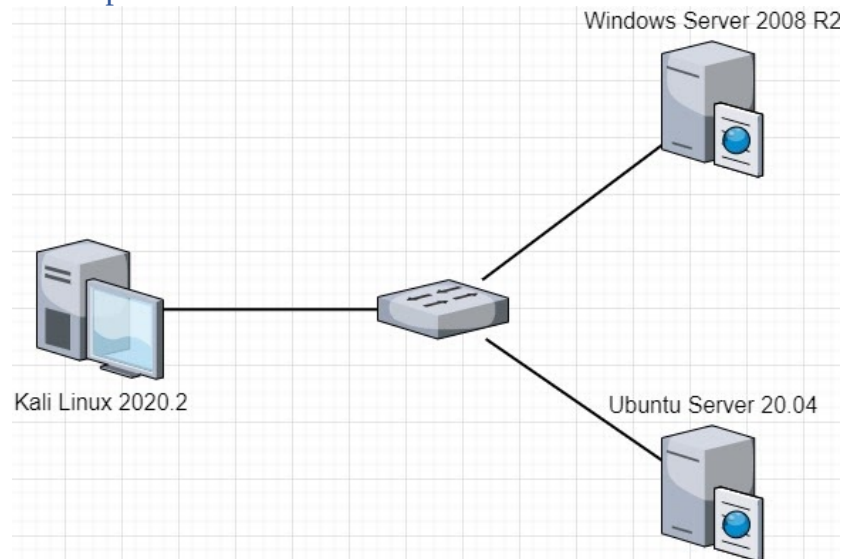


Figure 8 : Representation of the network topology

Figure 1 illustrates the network topology that was used for the creation of the network dataset. This network was private and separated from the internet. The network topology includes three entities in total. These entities are i) Kali Linux 2020.2, ii) Ubuntu Server 20.04, iii) Windows Server 2008 R2. Their use and purpose are thoroughly explained below.

#### 4.1.1 Windows Server 2008 R2

Windows Server 2008 R2, which is a product of the Metasploitable platform (see 3.4.3 Metasploitable 3), acted as a “victim” virtual machine. This machine is already vulnerable to a great number of vulnerabilities. Out of all the vulnerabilities that are already built-in, three were utilized. Those were mapped to the CVEs i) CVE-2014-3120<sup>24</sup>, ii) CVE-2015-8249<sup>25</sup>, and iii) CVE-2016-1209<sup>26</sup>. The afore-mentioned CVEs are Remote File Inclusion type vulnerabilities. Moreover, an SSH Server service was employed that was vulnerable to an SSH Bruteforce attack (CVE-2001-0553<sup>27</sup>). Also, for the evaluation process, a network packet capture component was deployed on this virtual machine.

<sup>24</sup> <https://www.cvedetails.com/cve/CVE-2014-3120>

<sup>25</sup> <https://www.cvedetails.com/cve/CVE-2015-8249>

<sup>26</sup> <https://www.cvedetails.com/cve/CVE-2016-1209>

<sup>27</sup> <https://www.cvedetails.com/cve/CVE-2001-0553/>



#### 4.1.2 Ubuntu Server 20.04

Just like Windows Server 2008 R2 that performs as a “victim” machine, Ubuntu Server 20.04 has the same role. On Ubuntu Server 20.04, an HTTP Web service was employed. That server was vulnerable to DoS Slow Loris attack (CVE-2007-6750<sup>28</sup>). Similarly, for the evaluation process, a network packet capture component was deployed on this virtual machine.

#### 4.1.3 Kali Linux 2020.2

The Kali Linux virtual machine has the role of the attacker. Through the use of Metasploit Framework’s Python3 API, three out of five attacks were automated. Those three attacks referred to the CVE vulnerabilities that Windows Server 2008 R2 has. Both SSH Bruteforce attack and DoS Slow Loris attack were automatically triggered from a custom python3 script. The SSH Bruteforce attack was initiated with the Hydra tool. DoS Slow Loris attack was initiated with a custom python3 script.

---

<sup>28</sup> <https://www.cvedetails.com/cve/CVE-2007-6750/>

## 4.2 Concept

This subsection presents all the architectural elements that compose the presented Situational Awareness framework. Furthermore, there is a brief description of each element's purpose and functionality within the framework. Figure 9 illustrates a high-level conceptual diagram of the presented framework. Nevertheless, some of the presented elements, such as the Wazuh server, the Wazuh agent and the Risk Assessment, are not within the scope of this thesis.

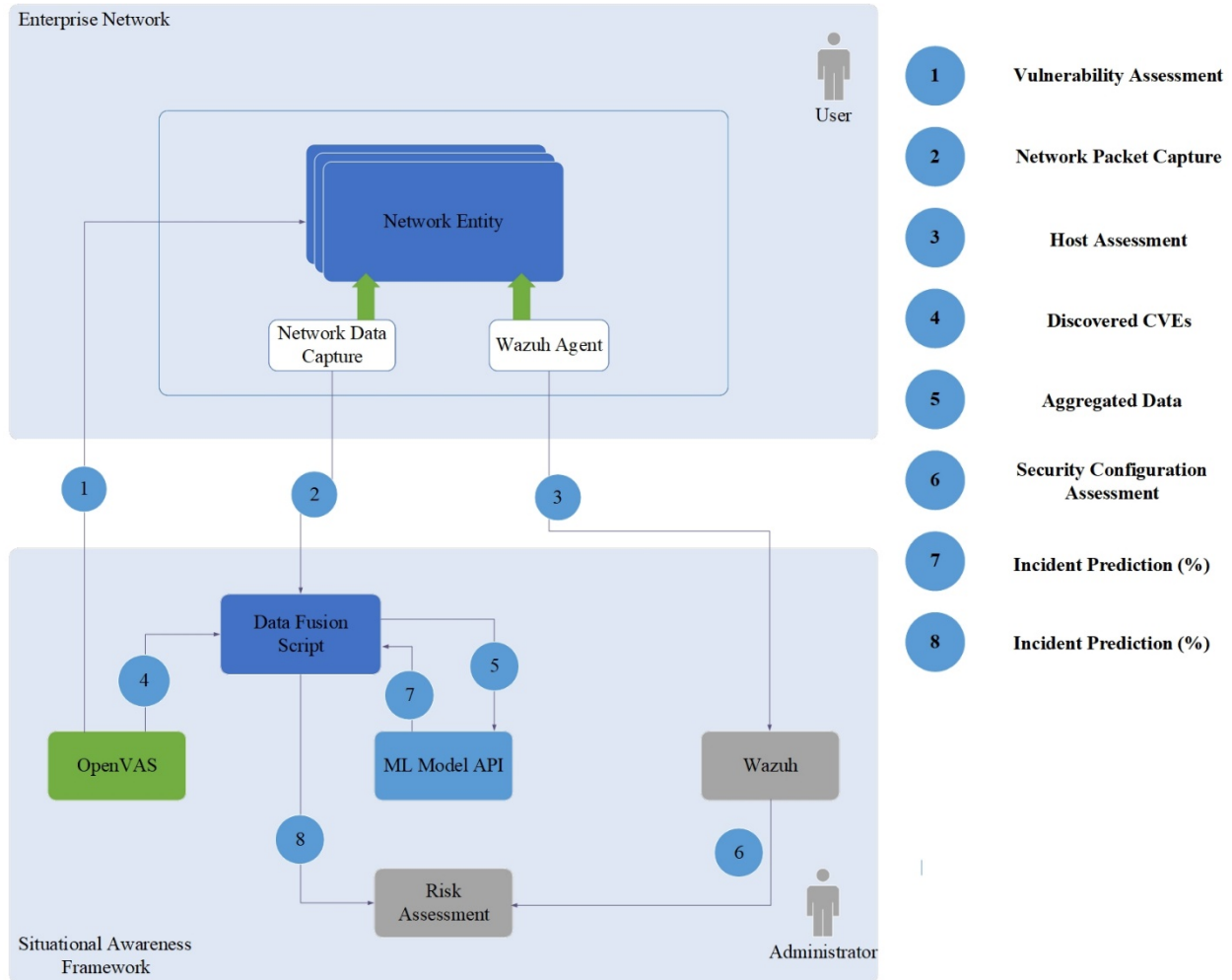


Figure 9: Conceptual Diagram

## 4.2.1 Architectural Elements

### 4.2.1.1 Network Data Capture

The Network Data Capture component is responsible for the network traffic capture procedure, from each network entity. The captured traffic is propagated to the Data Fusion script for further processing.

### 4.2.1.2 Wazuh

The Wazuh server produces the final Host Assessment Report and propagates the Security Configuration Assessment score to a Risk Assessment component. The Wazuh server component is out of the scope of this thesis.

### 4.2.1.3 Wazuh Agent

The Wazuh Agent, which is deployed on every network entity, propagates the entity's system logs to the Wazuh Server, for the server to generate the host assessment report of the entity.

### 4.2.1.4 OpenVAS

Initially, OpenVAS performs the vulnerability assessment. Afterwards, it propagates the CVE entries that it has discovered from the network entity to the Data Fusion script. This process is performed automatically through the use of the RESTful API that we have developed.

### 4.2.1.5 Data Fusion Script

The Data Fusion script aggregates and fuses the received data. As a result, it is responsible for the enhancement of the data. Consequently, the enhanced data is sent to the Machine Learning Model's API, as a POST Request. After the machine learning model has processed the data, the API will then return the incident prediction score to the Data Fusion script. Finally, the incident prediction score is sent to the Risk Assessment module for the final assessment.

### 4.2.1.6 Machine Learning Model API

TensorFlow's service, TensorFlow Serving, hosts the produced model of the trained Neural Network. Through a REST API, the model can make predictions on the enhanced data that the Data Fusion script provides. After the calculation of the incident prediction score by the model, the REST API returns the score to the Data Fusion script.

### 4.2.1.7 Risk Assessment

The Risk Assessment component receives the incident prediction score from the Data Fusion script and the Security Configuration Assessment score from the Wazuh server. Thereafter, it produces a risk percentage for the entity. The Risk Assessment component is out of the scope of this thesis.

### 4.3 Dataset Creation

The need for a dataset arose because most of the free network traffic datasets have been under heavy modification before they are published. This means that a part of the attack's network traffic is removed or filtered. This implies that the actual training of the model will not correspond to the actual attack. For that reason, a new network traffic dataset was created, wherein no information is excluded from the dataset, thus, providing more realistic results.

Initially, we executed and captured each attack individually. While the automated attacks were performed, we captured the network traffic data through the " victims' " virtual machines. Afterwards, we converted each raw network traffic batch that was captured for each attack, into network flow with 83 features and one label. Then, we labelled each network flow after the attack that corresponded to it. Afterwards, we concatenated those different batches to one single network flow dataset. The network flow dataset was then assessed by Weka which then concluded to a dimensionality reduction of 24 features. Thus, the network dataset concluded at 59 features.

The next step was to enhance the network dataset with data from the vulnerability assessment. For each vulnerability that matched one or more CVE entries, we added one more feature in the network dataset. Each feature could have a numerical value of 1 or 0 for each network flow. If the network flow that represented a specific attack was mapped with the CVE entry the feature would have the value 1. In any other case, it would have 0. There were five vulnerabilities, each one mapped to a CVE entry, thus, the enhanced network dataset was concluded at 64 features (Table 1). Then the final enhanced dataset was again assessed by Weka that resulted in that dimensionality reduction was not required. Since the dataset has more than 2 distinct labels, it lies upon the category of multiclass classification.

For the training and test sets to be representative of the overall distribution of data, to improve the quality and predictive performance, and to avoid overfitting the model, each dataset was shuffled beforehand [40] [41].



<b>Feature Name</b>	<b>Description</b>
<i>Flow duration</i>	Duration of the flow in Microsecond
<i>total Fwd Packet</i>	Total packets in the forward direction
<i>total Bwd packets</i>	Total packets in the backward direction
<i>total Length of Fwd Packet</i>	Total size of packet in forward direction
<i>total Length of Bwd Packet</i>	Total size of packet in backward direction
<i>Fwd Packet Length Min</i>	Minimum size of packet in forward direction
<i>Fwd Packet Length Max</i>	Maximum size of packet in forward direction
<i>Fwd Packet Length Mean</i>	Mean size of packet in forward direction
<i>Fwd Packet Length Std</i>	Standard deviation size of packet in forward direction
<i>Bwd Packet Length Max</i>	Maximum size of packet in backward direction
<i>Bwd Packet Length Mean</i>	Mean size of packet in backward direction
<i>Bwd Packet Length Std</i>	Standard deviation size of packet in backward direction
<i>Flow Bytes/s</i>	Number of flow bytes per second
<i>Flow Packets/s</i>	Number of flow packets per second
<i>Flow IAT Mean</i>	Mean time between two packets sent in the flow
<i>Flow IAT Std</i>	Standard deviation time between two packets sent in the flow
<i>Flow IAT Max</i>	Maximum time between two packets sent in the flow
<i>Flow IAT Min</i>	Minimum time between two packets sent in the flow
<i>Fwd IAT Min</i>	Minimum time between two packets sent in the forward direction
<i>Fwd IAT Max</i>	Maximum time between two packets sent in the forward direction
<i>Fwd IAT Mean</i>	Mean time between two packets sent in the forward direction
<i>Fwd IAT Std</i>	Standard deviation time between two packets sent in the forward direction
<i>Fwd IAT Total</i>	Total time between two packets sent in the forward direction
<i>Bwd IAT Min</i>	Minimum time between two packets sent in the backward direction
<i>Bwd IAT Max</i>	Maximum time between two packets sent in the backward direction
<i>Bwd IAT Mean</i>	Mean time between two packets sent in the backward direction
<i>Bwd IAT Std</i>	Standard deviation time between two packets sent in the backward direction
<i>Bwd IAT Total</i>	Total time between two packets sent in the backward direction
<i>Bwd PSH Flags</i>	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
<i>Fwd Header Length</i>	Total bytes used for headers in the forward direction
<i>Bwd Header Length</i>	Total bytes used for headers in the backward direction
<i>FWD Packets/s</i>	Number of forward packets per second
<i>Bwd Packets/s</i>	Number of backward packets per second
<i>Packet Length Max</i>	Maximum length of a packet
<i>Packet Length Mean</i>	Mean length of a packet
<i>Packet Length Std</i>	Standard deviation length of a packet
<i>Packet Length Variance</i>	Variance length of a packet
<i>FIN Flag Count</i>	Number of packets with FIN

<i>SYN Flag Count</i>	Number of packets with SYN
<i>PSH Flag Count</i>	Number of packets with PUSH
<i>ACK Flag Count</i>	Number of packets with ACK
<i>down/Up Ratio</i>	Download and upload ratio
<i>Average Packet Size</i>	Average size of packet
<i>Fwd Segment Size Avg</i>	Average size observed in the forward direction
<i>Bwd Segment Size Avg</i>	Average number of bytes bulk rate in the backward direction
<i>Subflow Fwd Packets</i>	The average number of packets in a sub flow in the forward direction
<i>Subflow Fwd Bytes</i>	The average number of bytes in a sub flow in the forward direction
<i>Subflow Bwd Packets</i>	The average number of packets in a sub flow in the backward direction
<i>Subflow Bwd Bytes</i>	The average number of bytes in a sub flow in the backward direction
<i>Bwd Init Win bytes</i>	The total number of bytes sent in initial window in the backward direction
<i>Fwd Act Data Pkts</i>	Count of packets with at least 1 byte of TCP data payload in the forward direction
<i>Active Min</i>	Minimum time a flow was active before becoming idle
<i>Active Mean</i>	Mean time a flow was active before becoming idle
<i>Active Max</i>	Maximum time a flow was active before becoming idle
<i>Active Std</i>	Standard deviation time a flow was active before becoming idle
<i>Idle Min</i>	Minimum time a flow was idle before becoming active
<i>Idle Mean</i>	Mean time a flow was idle before becoming active
<i>Idle Max</i>	Maximum time a flow was idle before becoming active
<i>Idle Std</i>	Standard deviation time a flow was idle before becoming active
<i>CVE_1</i>	It indicates that a CVE is present.
<i>CVE_2</i>	It indicates that a CVE is present.
<i>CVE_3</i>	It indicates that a CVE is present.
<i>CVE_4</i>	It indicates that a CVE is present.
<i>CVE_5</i>	It indicates that a CVE is present.
<i>Label</i>	The label of the flow.

Table 1 : Enhanced Dataset Features. (64)

Table 1

#### 4.4 Machine Learning Model Development

To compare the two afore-mentioned datasets, we came across the necessity of a base model. The afore-mentioned model will then be used for both datasets to be trained and tested. Below we present and explain the whole process that was applied to achieve this.

On each iteration of the process, a change in the parameters was made. Those changes were based on the fact that we have a dataset with more than two classes (labels), in our case we have five classes, one for each attack simulated. This means that we have a multiclass classification problem and based on that we have some recommended options for each parameter according to the literature [42] [43] [44] [45].

As stated in the bibliography, the activation function of each layer is one of the most significant parameters while training a neural network. Rectified Linear Unit (ReLU) for example is the first activation function that someone has to experiment with, to produce high accuracy results. On iterations one to seven, ReLU was used as the activation function.

Some parameters did not change throughout development. The output layer of the model had the same activation function throughout the whole process. That activation function was SoftMax. Moreover, the loss function also remained the same, which was the sparse categorical cross-entropy<sup>29</sup> function. According to the literature [46], the sparse categorical cross-entropy, and SoftMax functions were chosen because they are the optimal choices for multiclass classification problems when the labels are encoded into integers. Also, the optimizer<sup>30</sup> Adam, which is a version of stochastic gradient descent, which is also recommended for multiclass classification [47], was always the same. Furthermore, the accuracy and loss diagrams must not have any extreme outliers. If this occurs, it indicates that the model is overfitting [48].

It should be noted that the training process of the neural model that produced the results discussed below, was performed with the normal network dataset, which contains the 59 features. To prove that a neural model, which is trained with an enhanced dataset, performs better, we must first have an already trained model, with a normal network dataset, to compare it with. Next, we briefly discuss what the results of each iteration were.

The results that we compared are the training and test loss and accuracy respectively. The loss rate is estimated by adding the sum of errors made both in training and testing sets. The result of this calculation identifies how well or how poorly the model behaves. Values closer to 0 indicate the best possible result. On the contrary, the accuracy rate of a machine learning model is estimated by how frequently the model produces a correct prediction, with values closer to 1 being the best.

During the first iteration of the training process, the parameters that were used are shown in Table 2 resulting in a training accuracy of 0.671 and a test accuracy of 0.669. In this iteration, we noticed a high value of test loss (6960.123), while the training loss was considerably lower (1.139). In the second iteration, we chose to change the number of hidden layers from 1 to 2 (total number of layers changed from 3 to 4) and kept the rest of the parameters the same.

---

<sup>29</sup> [https://keras.io/api/losses/probabilistic\\_losses/](https://keras.io/api/losses/probabilistic_losses/)

<sup>30</sup> <https://keras.io/api/optimizers/>



Results presented a higher value of train and test accuracy (both of them were 0.730 while in the first iteration were 0.671 and 0.669 respectively) and also a significant decrease in both train and test loss (0.581 from 1.139 and 4057.63 from 6960.123 respectively). Continuing to the third iteration, we chose to use one hidden layer (3 in total) like in the first iteration and instead decrease the number of epochs (from 120, in the first and second iteration, to 70). Regarding this change, we observed an increase in accuracy, both in train and test (from 0.671 to 0.717 and from 0.669 to 0.731 respectively, compared to the first iteration) and a significant decrease in test loss (from 6960.123 to 886.701), although the training loss appeared to increase, compared to the first iteration (from 1.139 to 589.973). However, both train and test accuracy increased more during the second iteration compared to the third.

Moreover, for the fourth iteration, we decided to increase the number of hidden layers again, from 1 to 2 (total number of layers: 4) compared to the third iteration, but we kept the number of epochs at 70 like in the third iteration. The results of this iteration were conflicting, as it was noted, not only a decrease in both train and test accuracy (0.609 from 0.717 and 0.613 from 0.731 respectively) but also in train and test loss (30.949 from 589.373 and 347.513 from 886.701 respectively). Also, it is worth mentioning here that both train and test accuracy levels during the fourth iteration, not only decreased, compared to the third iteration, but they lowered even compared to the first iteration. Based on Figure 10, we concluded that the random and rather extreme outliers on the accuracy diagram of the fourth iteration indicate signs of overfitting.

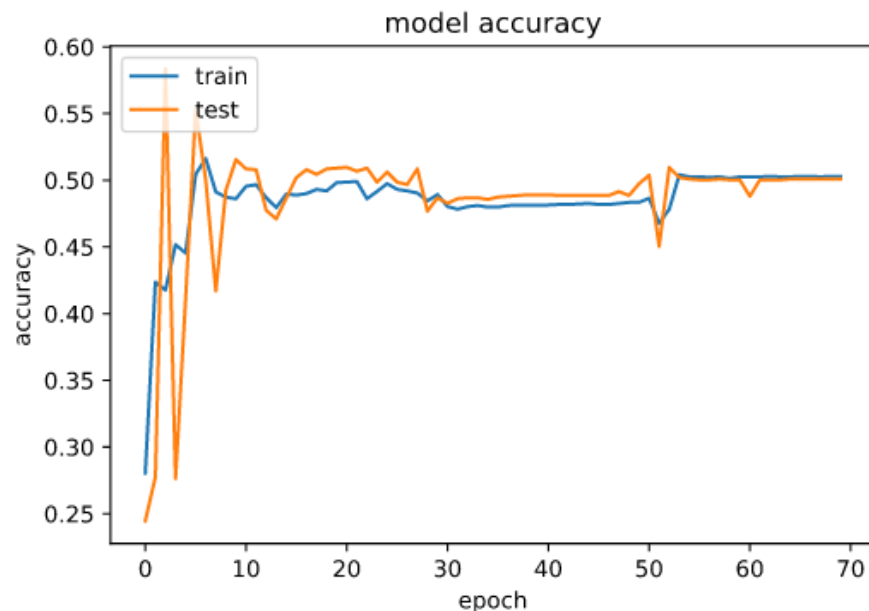


Figure 10 : 4th Iteration's Model Accuracy

Proceeding to iteration five, we kept everything as in the fourth iteration but we selected to use 10 nodes in the input layer rather than 20 nodes that we used in all preceding iterations, to see whether the overfitting problem will be resolved. As an outcome, a great increase in both train and test accuracy (0.904 from 0.609 and 0.915 from 0.613 respectively) was observed, and also a great decrease in test loss (13.548 from 347.513). However, train loss appeared to slightly increase, compared to the results of the fourth iteration (36.085 from 30.949).

In the sixth iteration, we chose to increase the batch size from 32 to 64, which resulted in an increase in accuracy levels (0.968 from 0.904 and 0.954 from 0.915), and a slight decrease in train loss. However, we also noticed a vast increase in test loss (from 13.548 to 3593.506). Those results prompted us to use the hyperbolic tangent function (Tanh), instead of the ReLU. In the seventh iteration, we used the same parameters, as in the sixth iteration, except the batch size that was changed back to 32. The results showed a slight decrease in accuracy levels, both train, and test (0.919 from 0.968 and 0.914 from 0.954 respectively). The mentionable outcome here is the major decrease of train and test loss (0.229 from 34.892 and 0.225 from 3593.506 respectively), a decrease a lot lower than in other iterations.

In our attempt to achieve better results in accuracy and loss levels, we tried to change the hidden layers back to 3, compared to 4 layers used in the seventh iteration, but, this eighth iteration, led to a decrease in accuracy and an increase in loss levels. Finally, in the ninth iteration, we decided to change the number of nodes in the input layer back to twenty, as in the very first iteration, while keeping everything else as in the eighth iteration and we got a training accuracy of 0.939, test accuracy of 0.945, train loss of 0.122 and test loss of 0.169.

Further combinations of different values of the parameters failed to result in a better outcome, than the one observed in the ninth iteration. This procedure led to the decision of maintaining these parameters, which resulted in the best outcome, to be trained and tested with the second (enhanced) dataset also.

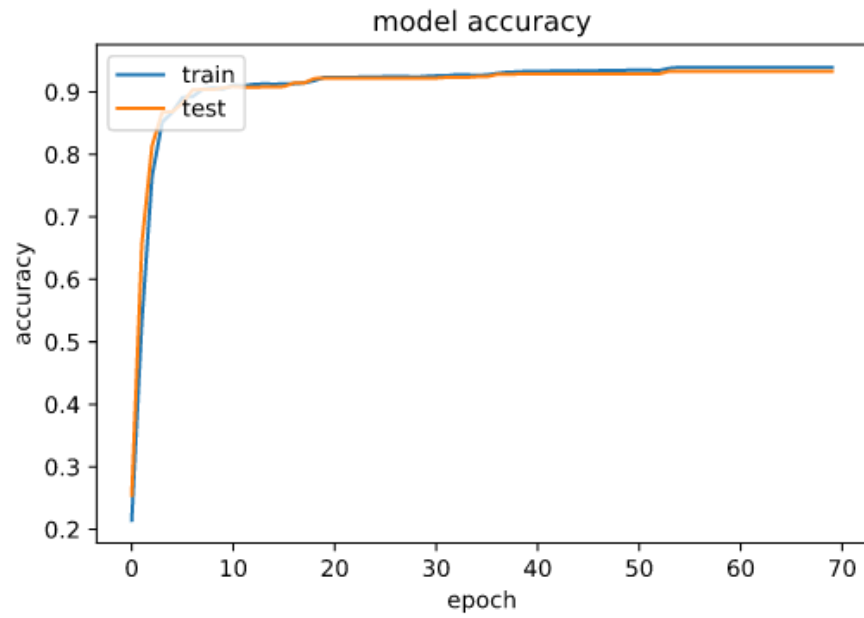


Figure 11 : 9th Iteration's Model Accuracy

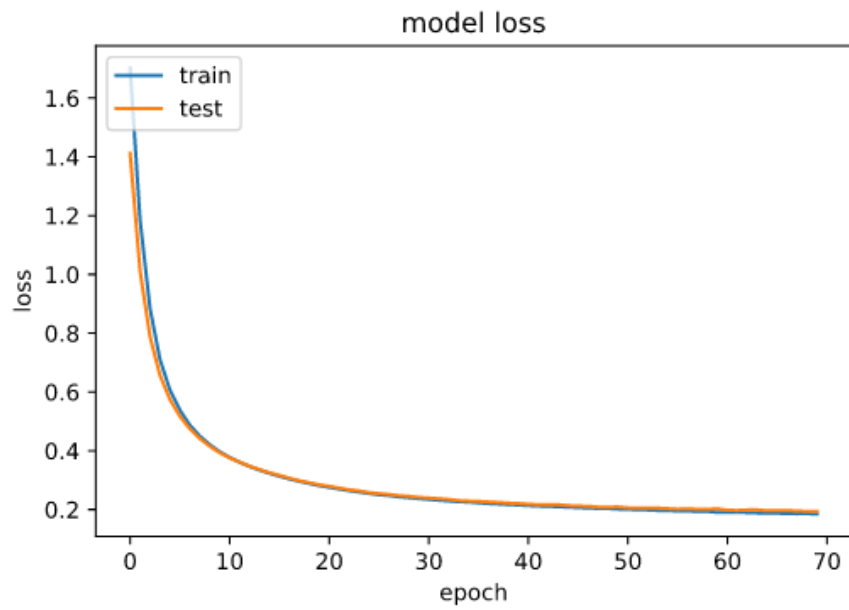


Figure 12 : 9th Iteration's Model Loss

<i>Experiment Iteration</i>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>
<i>No. of layers</i>	3	4	3	4	4	4	4	3	3
<i>Number of nodes of the input layer</i>	20	20	20	20	10	10	10	10	20
<i>Activation Functions</i>	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	Tanh	Tanh	Tanh
<i>Epochs</i>	120	70	70	70	70	70	70	70	70
<i>Batch Size</i>	32	32	32	32	32	64	32	32	32
<i>Train Accuracy</i>	<b>0.671</b>	<b>0.730</b>	<b>0.717</b>	<b>0.609</b>	<b>0.904</b>	<b>0.968</b>	<b>0.919</b>	<b>0.731</b>	<b>0.939</b>
<i>Train Loss</i>	<b>1.139</b>	<b>0.581</b>	<b>589.323</b>	<b>30.949</b>	<b>36.085</b>	<b>34.892</b>	<b>0.229</b>	<b>0.658</b>	<b>0.182</b>
<i>Test Accuracy</i>	<b>0.669</b>	<b>0.730</b>	<b>0.731</b>	<b>0.613</b>	<b>0.915</b>	<b>0.954</b>	<b>0.914</b>	<b>0.713</b>	<b>0.933</b>
<i>Test Loss</i>	<b>6960.123</b>	<b>4057.630</b>	<b>886.701</b>	<b>347.513</b>	<b>13.548</b>	<b>3593.506</b>	<b>0.225</b>	<b>0.693</b>	<b>0.193</b>

Table 2 : Parameters changed throughout the iterations, and Accuracy and Loss produced

Taking everything into consideration, in our attempt to find the best model to be trained, with the highest train and test accuracy, while retaining the lowest train and test loss, we noticed that when we changed the activation function from ReLU, which was used in the first six iterations, to hyperbolic tangent function (Tanh), a considerable increase in both train and test accuracy, and a decrease in train and test loss occurred. According to the bibliography, ReLU is the activation function that is used more often and is also the one that is expected to operate better among most situations. However, considering each neural network's specificities and also one's unique model architecture, it might not always be the case. Furthermore, we are aware that the Tanh suits better for neural networks that are developed to make predictions, and possibly that is the reason why this activation function works better in our use case.

Finally, when the best model was obtained through those iterations, we also trained it with the enhanced dataset.

The model that is trained with the normal network traffic dataset, will be from now on referred to as "Model #1". Similarly, the model that is trained on the network traffic dataset with enhanced features, will be referred to as "Model #2".

The outcome of the comparison of the two differently trained models is shown in Table 3.

	<i>Trained Model with the normal dataset (Model #1)</i>	<i>Trained Model with the enhanced dataset (Model #2)</i>
<b>Train Accuracy</b>	0.939	<b>0.953</b>
<b>Train Loss</b>	0.182	<b>0.118</b>
<b>Test Accuracy</b>	0.933	<b>0.951</b>
<b>Test Loss</b>	0.193	<b>0.118</b>

Table 3 : Comparison of accuracy and loss between the two different trained models

Figure 13 and Figure 14, showcase the accuracy and loss diagrams of Model #2.

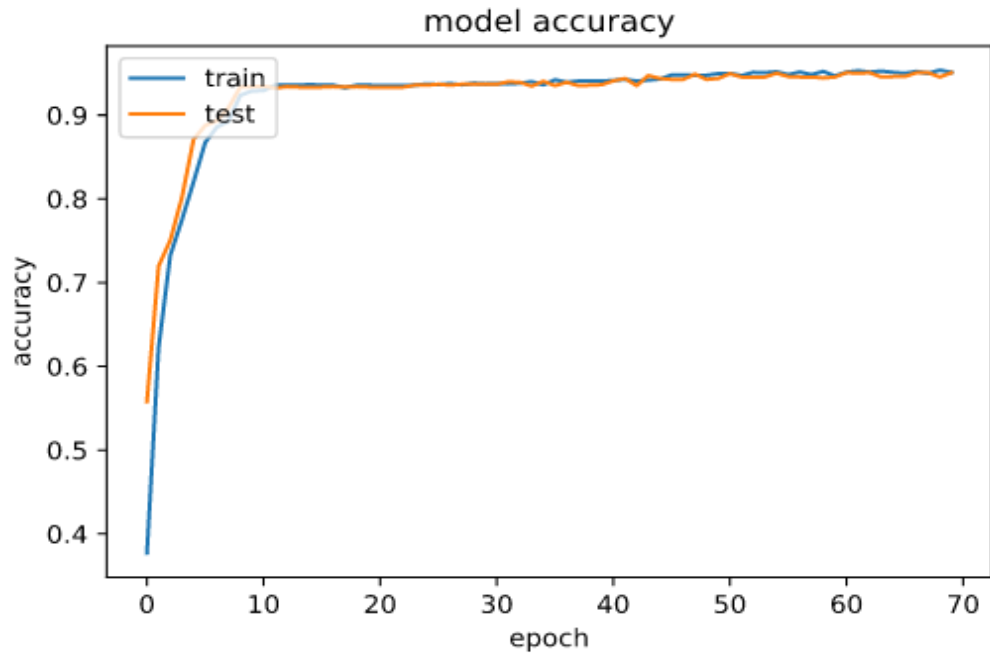


Figure 13 : Model #2 Accuracy

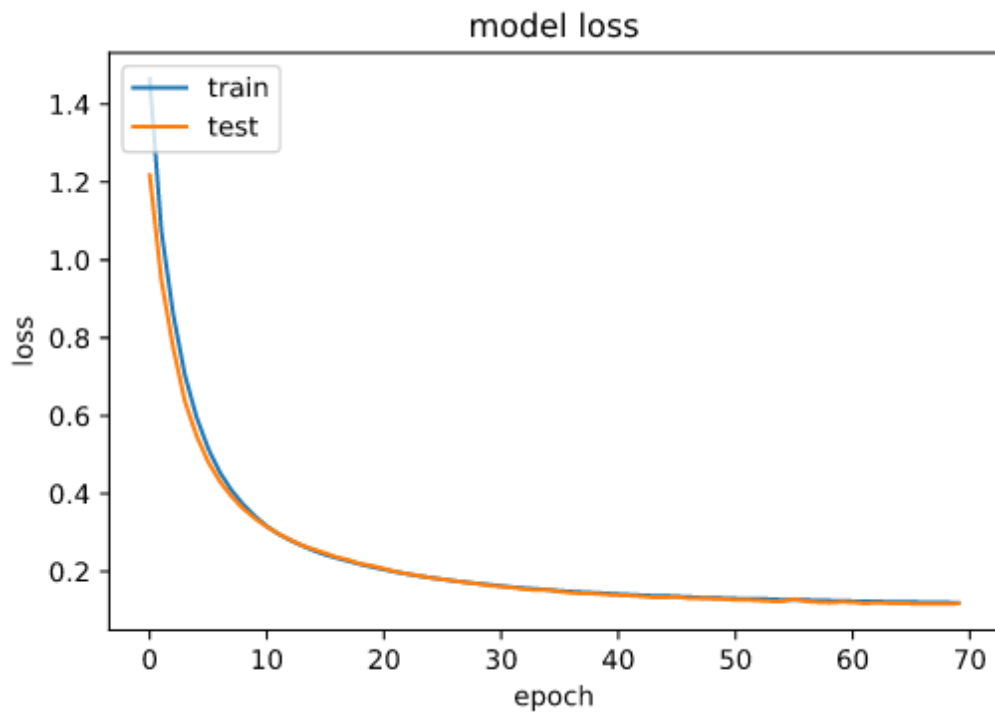


Figure 14 : Model #2 Loss

As presented in Table 4, train accuracy for Model #2 was measured as 0.953, whereas the train accuracy levels of Model #1 appear to be a little bit lower, at 0.939. The test accuracy level of Model #2 is also increased compared to Model #1, 0.951 and 0.933 respectively. We also observe a decrease in both train and test loss regarding Model #2 in comparison with Model #1. 0.118 and 0.182 for train loss and 0.118 and 0.193 for test loss respectively. In consideration of the foregoing, Model #2, showed an increase in both train and test accuracy by 1.6%, as well as a decrease in train and test loss by 7% compared to Model #1.

Figure 13, indicates that because both the training and testing process of Model #2 does not show any extreme outliers, the model appears to have a good fit learning curve. A good fitting learning curve is a curve in which the plot of training loss gradually declines until it becomes stable. Also, a learning curve serves a good fit when the plot of test loss showcases a small difference, or otherwise gap, comparing to training loss, regarding the starting point of the curves and equally declines gradually until it becomes stable. Figure 14 indicates that not only the learning curve decreases to a point of stability, but we can also notice a small gap between the train and test loss.

#### 4.5 Use Case

In this section, we will look into the general use case of the proposed framework.

A vast amount of data is coming in and out of an enterprise network. Every network administrator must be aware of this data and make sense out of it. By harnessing those data and by distinguishing the benign from malicious activity, a network administrator can protect an enterprise from cyber threats while preventing financial losses and sensitive information leaks.

As showcased in Figure 15, the Situational Awareness Framework will start by assessing a network entity. Then, the network traffic of the entity is captured. After that, the Wazuh Agent makes a host assessment. Then, OpenVAS performs a vulnerability assessment on the entity. Finally, the Wazuh agent sends its results to the Wazuh server. Then the network packet capture component and OpenVAS send their results on the Data Fusion script. The network packet capture component sends the raw network traffic data, while OpenVAS sends its discovered CVE entries. The Data Fusion script converts the raw network traffic data to network flows, and then adds another feature to them, according to the number of CVEs discovered on the entity. Each CVE is added as a new feature with a numeric value of 1. Then, the Data Fusion script makes a POST request to the trained Machine Learning Model API which, in turn, returns the Incident Prediction score. After that, the Security Configuration Assessment Score, OpenVAS, and the Data Fusion script scores are being sent to the Risk Assessment Component.

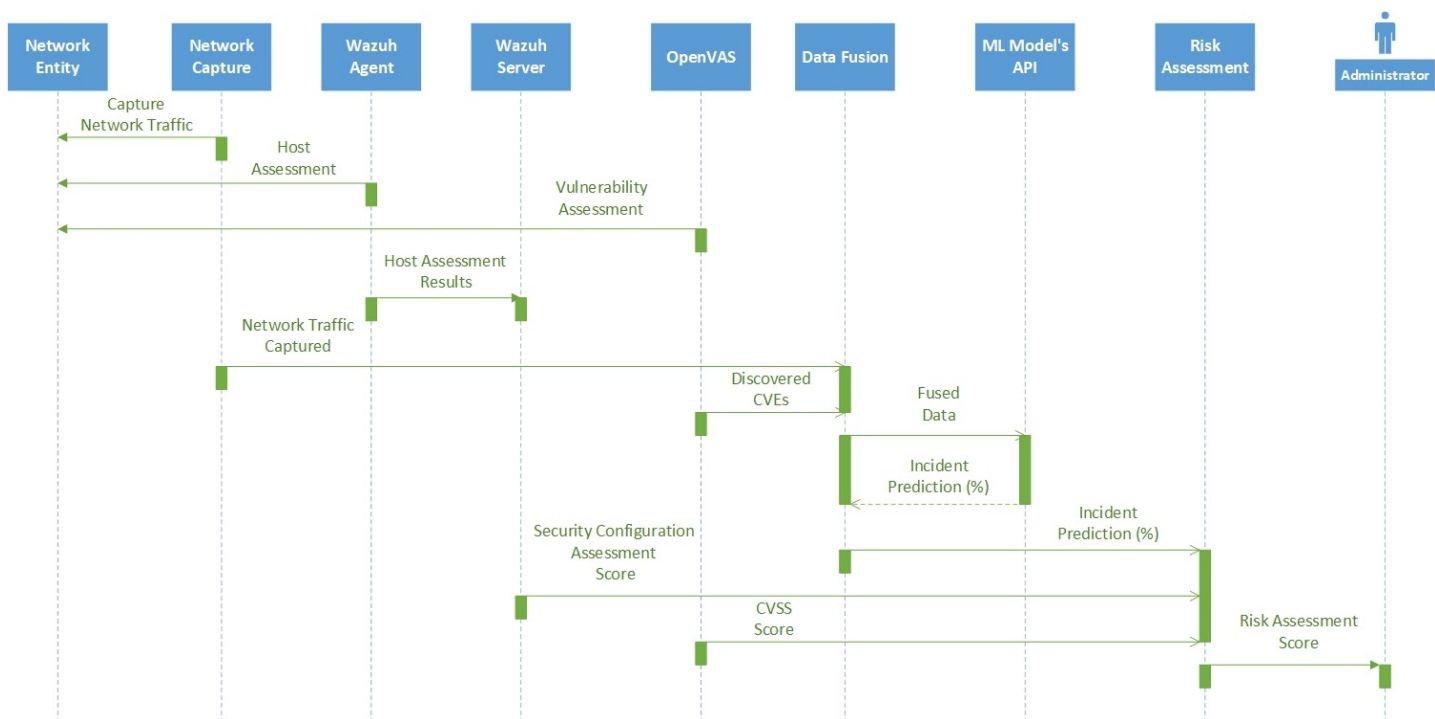


Figure 15 : Situational Awareness Framework Sequence Diagram

## 5. Evaluation

### 5.1 Aim

The scope of this evaluation is to assess the efficacy and overall precision of the neural network of our proposed ML-powered Situational Awareness framework. The efficacy is defined as the total number of false-positive predictions (Formula 1), and precision is defined as the number of correctly predicted attacks out of all the performed attacks, which can be estimated by dividing the sum of correct predictions by the sum of total predictions, as demonstrated in Formula 2.



$$Efficacy = AllPredictions - CorrectPredictions$$

*Formula 1*

$$Precision = \frac{Correct\ Predictions}{AllPredictions}$$

*Formula 2*

## 5.2 Method

The evaluation process is a two-fold procedure. For the first phase, we assess the efficacy and precision of the ML model, trained with the enhanced dataset. For the second phase, we assess the same values for the M model, which was trained with the plain dataset.

For each phase, a network traffic sniffer and the OpenVas platform were utilized, to capture live network flows, and system vulnerabilities, expressed in the CVE description format, respectively (Table 4). A python-based fusion script gathers and fuses the heterogeneous data and transforms it into a format acceptable (NumPy arrays) by the ML model. The transformed input array was fed to the ML model, which produced a prediction on what will happen in the next moments (attack or no attack).

<i>Service</i>	<i>CVE Entry</i>	<i>Vulnerable type of attack</i>
Elasticsearch	CVE-2014-3120	Remote File Inclusion (RFI) Attack - <b>RFI Attack #1</b>
ManageEngine Remote Desktop 9	CVE-2015-8249	RFI Attack - <b>RFI Attack #2</b>
WordPress NinjaForms 2.9.42	CVE-2016-1209	RFI Attack - <b>RFI Attack #3</b>
Open 22 Port with OpenSSH Server Service	CVE-2001-0553	SSH Bruteforce Attack - <b>Attack #4</b>
Vulnerable HTTP Web Server	CVE-2007-6750	DoS Slow Loris - <b>Attack #5</b>

*Table 4 : Vulnerabilities and their corresponding CVE entries and attack types*

## 5.3 Variables

### 5.3.1 Dependent Variables

The dependent variables of this evaluation experiment are the precision of each prediction (%), and the total number of false-positive predictions for each differently trained model. The values collected for each model were assessed and compared with each other. This allowed us to gain a more detailed assessment, on the efficacy and precision of each respective model.

### 5.3.2 Independent Variables

For this evaluation experiment, we required only one independent variable, the type of attack. During the experiment, we performed all the attacks the models were trained for.

### 5.3.3 Fixed Variables

The fixed variables for this experiment were the total duration for each respective attack. In more detail, the DoS Slow Loris and SSH Bruteforce attacks that are time-based lasted for two minutes each. The three remote file inclusion (RFI) attacks were repeated forty times each

### 5.4 Prediction

The assumption that we are trying to prove is that Model #2, will generate a greater precision rate on an identical cyber incident situation than Model #1. Moreover, we expect the false-positive predictions for Model #2 to be zero (0).

### 5.5 Results

Both Table 5 and Figure 16 illustrate the difference in prediction precision between the two models.

	RFI Attack #1	RFI Attack #2	RFI Attack #3	Attack #4	Attack #5
<b>Model #1 Accuracy</b>	86.420	<b>66.807</b>	76.923	96.011	91.863
<b>Model #2 Accuracy</b>	<b>95.556</b>	65.966	<b>77.198</b>	<b>96.065</b>	<b>92.827</b>

Table 5 : Model's Predictions on live attacks

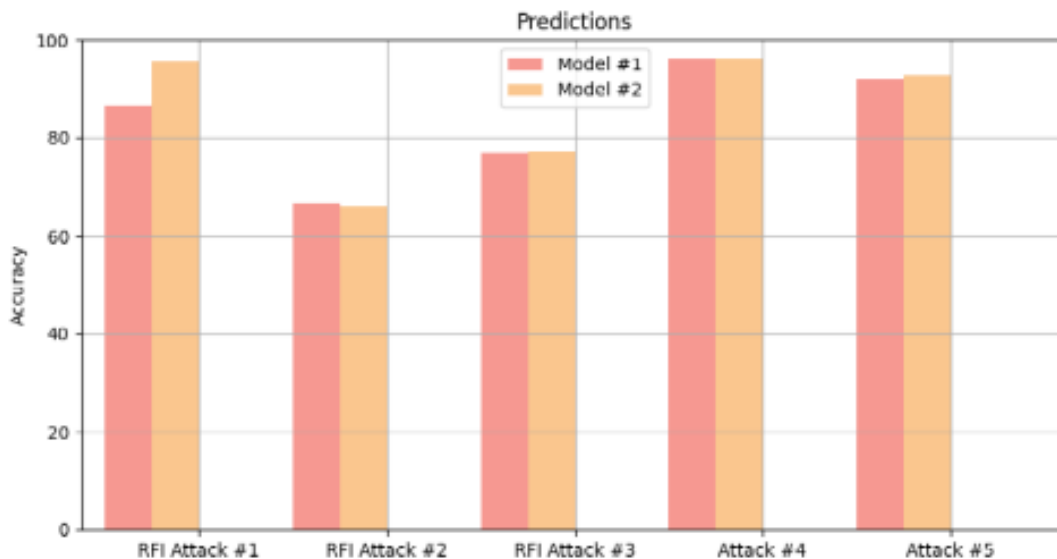


Figure 16 : Graph representation of Models' Predictions on live attacks

## 5.6 Discussion

Both models are compared in Table 5, regarding their precision on predicting each of the attacks mentioned in subsection 5.2 Method. The first row represents the precision values of Model #1 and the second row represents the precision values of Model #2.

Results signify that Model #2 produces more accurate prediction results on four out of five attacks. Model #1 has an average precision rate of 83.6%, and Model #2 has an average precision rate of 85.5%. Precision is increased by almost 2% for Model #2 in comparison to Model #1. The graphical comparison of the two models can also be shown in Figure 16.

To summarize, Model #2 performed better than Model #1, considering the prediction of the attacks that were replicated. The precision levels of Model #2 were higher than Model #1.

Model #2 performed better in the prediction of four out of five of the attacks, although when attack #3 was replicated, the prediction of both models was not significantly high. Even in this case though, our prediction is confirmed because the second model performed better than the first. In our research, there was only one attack (#2) that was not predicted accurately enough by the second model to surpass the first model. It is worth mentioning that attack #2 also, like attack #3, was not predicted accurately enough and that may have been caused because three out of five attacks were the same type.

Furthermore, the results of this experiment were somewhat controversial. In general, Model #2 performs better in most of the cases. Nevertheless, in one particular case it performs exceptionally, and in another case, Model #2 is outperformed by Model #1.

In more detail, as mentioned in subsection 4.3 Dataset Creation, Weka did not result in any dimensionality reduction after we added the extra features. Those extra features, assisted to the overall better prediction. However, the largest part of the dataset, consists of network traffic related data. This means that the network traffic has a greater impact on the performance of our model (Model #2). Therefore, taking under consideration that some attacks produce a larger network footprint than others, it is understandable that our model's behavior will be dependent on the type of attack, in respect to the network footprint it produces. This is the reason that, in one case, even if Model #2 performed better, the precision did not surpass 80%. Moreover, in one case we observe that Model #2 was outperformed by Model #1, and the overall precision was well under 70%. This occurrence was a result of an attack that did not leave a noteworthy network footprint. Additionally, the extended dimensionality of the model did not act in favor of the prediction.

Furthermore, we observe that in two occurrences Model #2 performs better than Model #1, and the overall precision is well over 95%. Once more, the attack produced a notable network footprint, and the extra dimensionality of the model helped the model to perform better.

## 5.7 Evaluation

The evaluation presented so far, showcases and compares the different efficiency a neural network model has on two different occasions. Firstly, when it is trained upon a network traffic dataset, and then when it is trained upon a heterogeneous-data dataset. The results were decisive for that comparison. Even so, the experiment's variables could be modified further for even better results. From our perspective, concerning the neural network model, more algorithms could be tested, to produce even higher prediction precision. Furthermore, Model #2 produces higher precision prediction on four out of five attacks, in comparison with Model #1. However, the method utilized for the creation of that dataset could be changed, since it now causes the model to perform better but if the total number of attacks increases, it may not perform as well. Further investigation is suggested for the creation of the fused heterogeneous-data dataset.

# 6. Conclusion

In this thesis, we proposed a framework that provides vulnerability, network, and host awareness to an administrator of an enterprise network. Our main focus on this research was the enhancement of the network awareness by utilizing a neural network that was trained upon an enhanced network traffic dataset. The purpose of the study was to confirm the proof of concept that a neural network produces higher accuracy predictions when trained upon an enhanced network dataset, in comparison with a neural network that was trained on a normal network traffic dataset and has the same architecture.

The results of our experiment showcase an average precision rate of 83.6% for Model #1, for the prediction of the replicated attacks, and an average precision rate of 85.5% for Model #2, for the prediction of the replicated attacks. In short, model#2 performed almost 2% better than model#1.

By examining the evaluation results, we deduced that the presented Situational Awareness Neural Network performs better in attacks that produce more profound footprint.

Two limitations require discussion. Firstly, each CVE entry that was found by the vulnerability assessment was added as a new feature on the network dataset. For this experiment, the CVE entries were only five but in a real case scenario, where more vulnerabilities might be present, this will end up to the addition of even more new features. This may require further dimensionality reduction for the neural network to be trained efficiently, which may result in the deprecation of some of the new features. Moreover, for the scope of this study, five different attacks were replicated, but three of them were of the same type (Remote File Inclusion - RFI). This probably led to the not as high as expected precision rates regarding attacks #2 and #3, because of the similarity of the network traffic data that they produced.

Further research should be conducted regarding the steps followed for the fusion of the data, to create a more refined way to produce the enhanced network dataset, so that the possible deprecation of newly added features would be avoided. Additionally, the architecture of the neural network must be revised. This would cause the attacks to be uniquely recognized, thus, to achieve higher precision rates when predicting the incoming attacks.

## 7. References

- [1] Y. Nikoloudakis, E. Pallis, G. Mastorakis, C. X. Mavromoustakis, C. Skianis, and E. K. Markakis, “Vulnerability assessment as a service for fog-centric ICT ecosystems: A healthcare use case,” *Peer-to-Peer Netw. Appl.*, vol. 12, no. 5, pp. 1216–1224, Sep. 2019, doi: 10.1007/s12083-019-0716-y.
- [2] Y. Nikoloudakis *et al.*, “A Fog-Based Emergency System for Smart Enhanced Living Environments,” *IEEE Cloud Comput.*, vol. 3, no. 6, pp. 54–62, Nov. 2016, doi: 10.1109/MCC.2016.118.
- [3] K. Karras *et al.*, “A Hardware Acceleration Platform for AI-Based Inference at the Edge,” *Circuits, Syst. Signal Process.*, vol. 39, no. 2, pp. 1059–1070, Feb. 2020, doi: 10.1007/s00034-019-01226-7.
- [4] Y. Nikoloudakis, E. Markakis, G. Mastorakis, E. Pallis, and C. Skianis, “An NFV-powered emergency system for smart enhanced living environments,” in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov. 2017, pp. 258–263, doi: 10.1109/NFV-SDN.2017.8169872.
- [5] Y. Nikoloudakis *et al.*, “Edge Caching Architecture for Media Delivery over P2P Networks,” in *2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, Sep. 2018, vol. 2018-Sept, pp. 1–5, doi: 10.1109/CAMAD.2018.8514935.
- [6] M. R. Endsley, “Toward a theory of situation awareness in dynamic systems,” *Hum. Factors*, vol. 37, no. 1, pp. 32–64, 1995, doi: 10.1518/001872095779049543.
- [7] T. Bass, “Multisensor Data Fusion for Next Generation Distributed Intrusion Detection Systems,” *Irish Natl. Symp.*, no. Id, pp. 24–27, 1999, doi: 10.1.1.51.1753.
- [8] T. Bass, “Intrusion detection systems and multisensor data fusion,” *Commun. ACM*, vol. 43, no. 4, pp. 99–105, 2000, doi: 10.1145/332051.332079.
- [9] T. Kokkonen, “Architecture for the Cyber Security Situational Awareness System,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9870 LNCS, 2016, pp. 294–302.
- [10] N. A. Giacobe, “Application of the JDL data fusion process model for cyber security,” in *Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2010*, Apr. 2010, vol. 7710, p. 77100R, doi: 10.1117/12.850275.
- [11] Y. Zhang, J. Zhang, and B. Zhang, “Visual Analysis of Cybersecurity Situational Awareness,” in *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, Oct. 2019, vol. 2019-Octob, pp. 685–688, doi:

- 10.1109/ICSESS47205.2019.9040716.
- [12] H. Shiravi, A. Shiravi, and A. A. Ghorbani, “A Survey of Visualization Systems for Network Security,” *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 8, pp. 1313–1329, Aug. 2012, doi: 10.1109/TVCG.2011.144.
- [13] L. Harrison and Aidong Lu, “The future of security visualization: Lessons from network visualization,” *IEEE Netw.*, vol. 26, no. 6, pp. 6–11, Nov. 2012, doi: 10.1109/MNET.2012.6375887.
- [14] H. K. Park, M. S. Kim, M. Park, and K. Lee, “Cyber situational awareness enhancement with regular expressions and an evaluation methodology,” in *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, Oct. 2017, pp. 406–411, doi: 10.1109/MILCOM.2017.8170859.
- [15] E. Doynikova and I. Kotenko, “CVSS-based Probabilistic Risk Assessment for Cyber Situational Awareness and Countermeasure Selection,” in *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, Apr. 2017, pp. 346–353, doi: 10.1109/PDP.2017.44.
- [16] T. Jirsik and P. Celeda, “Cyber Situation Awareness via IP Flow Monitoring,” in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, Apr. 2020, pp. 1–6, doi: 10.1109/NOMS47738.2020.9110327.
- [17] T. Jirsik and P. Celeda, “Toward real-time network-wide cyber situational awareness,” in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, Apr. 2018, pp. 1–7, doi: 10.1109/NOMS.2018.8406166.
- [18] X. W. Liu, H. Q. Wang, Y. Liang, and J. B. Lai, “Heterogeneous multi-sensor data fusion with multi-class support vector machines: Creating network security situation awareness,” *Proc. Sixth Int. Conf. Mach. Learn. Cybern. ICMLC 2007*, vol. 5, no. August, pp. 2689–2694, 2007, doi: 10.1109/ICMLC.2007.4370604.
- [19] X. Liu, H. Wang, J. Lai, Y. Liang, and C. Yang, “Multiclass Support Vector Machines Theory and Its Data Fusion Application in Network Security Situation Awareness,” in *2007 International Conference on Wireless Communications, Networking and Mobile Computing*, Sep. 2007, pp. 6343–6346, doi: 10.1109/WICOM.2007.1557.
- [20] L. Saitta, “Support-Vector Networks,” vol. 297, pp. 273–297, 1995, doi: 10.1007/BF00994018.
- [21] H. Wang, X. Liu, J. Lai, and Y. Liang, “Network security situation awareness based on heterogeneous multi-sensor data fusion and neural network,” in *Second International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2007)*, Aug. 2007, pp. 352–359, doi: 10.1109/IMSCCS.2007.15.
- [22] D. Svozil, V. Kvasnička, and J. Pospíchal, “Introduction to multi-layer feed-forward neural networks,” in *Chemometrics and Intelligent Laboratory Systems*, Nov. 1997, vol.

- 39, no. 1, pp. 43–62, doi: 10.1016/S0169-7439(97)00061-0.
- [23] M. L. Mathews, P. Halvorsen, A. Joshi, and T. Finin, “A collaborative approach to situational awareness for cybersecurity,” *Collab. 2012 - Proc. 8th Int. Conf. Collab. Comput. Networking, Appl. Work.*, pp. 216–222, 2012, doi: 10.4108/icst.collaboratecom.2012.250794.
- [24] J. UNDERCOFFER, A. JOSHI, T. FININ, and J. PINKSTON, “Using DAML+OIL to classify intrusive behaviours,” *Knowl. Eng. Rev.*, vol. 18, no. 3, pp. 221–241, Sep. 2003, doi: 10.1017/S0269888904000049.
- [25] J. Undercoffer, A. Joshi, and J. Pinkston, “Modeling Computer Attacks: An Ontology for Intrusion Detection,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2820, 2003, pp. 113–135.
- [26] H. Wang *et al.*, “Research on Network Security Situation Assessment and Quantification Method Based on Analytic Hierarchy Process,” *Wirel. Pers. Commun.*, vol. 102, no. 2, pp. 1401–1420, 2018, doi: 10.1007/s11277-017-5202-3.
- [27] L. A. Zadeh, “Simple View of the Dempster-Shafer Theory of Evidence and Its Implication for the Rule of Combination.,” *AI Mag.*, vol. 7, no. 2, pp. 85–90, 1986.
- [28] S. K. Srivastava, Y. K. Sharma, and S. Kumar, “Characteristics categorization dataset KDD Cup’99,” *AIP Conf. Proc.*, vol. 2142, no. August, 2019, doi: 10.1063/1.5122494.
- [29] Y. Yao, Y. Sun, Z. Liu, X. Meng, and Z. Liu, “A Data Fusion Framework of Multi-Source Heterogeneous Network Security Situational Awareness Based on Attack Pattern,” *J. Phys. Conf. Ser.*, vol. 1550, p. 062025, May 2020, doi: 10.1088/1742-6596/1550/6/062025.
- [30] E. Markakis, Y. Nikoloudakis, E. Pallis, and M. Manso, “Security Assessment as a Service Cross-Layered System for the Adoption of Digital, Personalised and Trusted Healthcare,” in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, Apr. 2019, pp. 91–94, doi: 10.1109/WF-IoT.2019.8767249.
- [31] T. Oladipupo, “Types of Machine Learning Algorithms,” in *New Advances in Machine Learning*, K. Pesek, Ed. InTech, 2010, p. 624.
- [32] S. K. Pal and S. Mitra, “Multilayer perceptron, fuzzy sets, classification,” *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 683–697, 1992.
- [33] A. G. Karegowda, A. S. Manjunath, G. Ratio, and C. F. Evaluation, “Comparative study of Attribute Selection Using Gain Ratio,” *Int. J. Inf. Technol. Knowl. Knowl. Manag.*, vol. 2, no. 2, pp. 271–277, 2010, [Online]. Available: <https://pdfs.semanticscholar.org/3555/1bc9ec8b6ee3c97c524f9c9ceee798c2026e.pdf>  
%0Ahttp://csjournals.com/IJITKM/PDF 3-1/19.pdf.



- [34] S. B. Kotsiantis and D. Kanellopoulos, “Data preprocessing for supervised learning,” *Int. J. ...*, vol. 1, no. 2, pp. 1–7, 2006, doi: 10.1080/02331931003692557.
- [35] W. McKinney, “Data Structures for Statistical Computing in Python,” in *Proceedings of the 9th Python in Science Conference*, 2010, vol. 1697900, no. Scipy, pp. 56–61, doi: 10.25080/Majora-92bf1922-00a.
- [36] S. van der Walt, S. C. Colbert, and G. Varoquaux, “The NumPy Array: A Structure for Efficient Numerical Computation,” *Comput. Sci. Eng.*, vol. 13, no. 2, pp. 22–30, Mar. 2011, doi: 10.1109/MCSE.2011.37.
- [37] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, “Characterization of Encrypted and VPN Traffic using Time-related Features,” in *Proceedings of the 2nd International Conference on Information Systems Security and Privacy*, 2016, no. Icissp, pp. 407–414, doi: 10.5220/0005740704070414.
- [38] A. H. Lashkari, G. D. Gil, M. S. I. Mamun, and A. A. Ghorbani, “Characterization of tor traffic using time based features,” *ICISSP 2017 - Proc. 3rd Int. Conf. Inf. Syst. Secur. Priv.*, vol. 2017-Janua, no. Cic, pp. 253–262, 2017, doi: 10.5220/0006105602530262.
- [39] S. Heron, “Denial of service: motivations and trends,” *Netw. Secur.*, vol. 2010, no. 5, pp. 10–12, May 2010, doi: 10.1016/S1353-4858(10)70056-8.
- [40] M. Gürbüzbalaban, A. Ozdaglar, and P. A. Parrilo, “Why random reshuffling beats stochastic gradient descent,” *Math. Program.*, Oct. 2019, doi: 10.1007/s10107-019-01440-w.
- [41] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, “Speeding Up Distributed Machine Learning Using Codes,” *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, Mar. 2018, doi: 10.1109/TIT.2017.2736066.
- [42] J. Brownlee, *Better Deep Learning: Train Faster, Reduce Overfitting, and Make Better Predictions*. Machine Learning Mastery, 2018.
- [43] J. Brownlee, *Master Machine Learning Algorithms: discover how they work and implement them from scratch*. Machine Learning Mastery, 2016.
- [44] J. Brownlee, “Supervised and unsupervised machine learning algorithms,” *Mach. Learn. Mastery*, vol. 16, no. 03, 2016.
- [45] J. Brownlee, *Deep learning with Python: develop deep learning models on Theano and TensorFlow using Keras*. Machine Learning Mastery, 2016.
- [46] R. Dunne and N. Campbell, “On the pairing of the Softmax activation and cross-entropy penalty functions and the derivation of the Softmax activation function,” *Proc. 8th Aust. Conf. Neural Networks*, pp. 1–5, 1997, doi: 10.1.1.49.6403.
- [47] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” *3rd Int. Conf.*

*Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–15, 2015.

- [48] J. Lever, M. Krzywinski, and N. Altman, “Model selection and overfitting,” *Nat. Methods*, vol. 13, no. 9, pp. 703–704, Sep. 2016, doi: 10.1038/nmeth.3968.