

Florida International University
Knight Foundation School of Computing and Information Sciences

sSoftware Engineering Focus

Final Deliverable

Project Title: Merging Intrusion Detection Systems

Team Members: Christopher Puig, Josue Camejo, Josue Naranjo, Dennis Martinez, Michael Simon.

Product Owner(s): Yanzhao Wu

Mentor(s): None

Instructor: Masoud Sadjadi

The MIT License (MIT)
Copyright (c) 2016 Florida International University

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Abstract

This document presents a comprehensive overview of the project "Merging Two Intrusion Detection Systems," which aims to enhance the accuracy and robustness of intrusion detection mechanisms by combining the strengths of two existing intrusion detection systems. In the ever-evolving landscape of cybersecurity threats, the need for reliable and efficient intrusion detection has become paramount to safeguard sensitive data and ensure the integrity of systems and networks.

The project's primary purpose is to achieve a higher level of accuracy and effectiveness in detecting and preventing intrusion attempts by merging two distinct intrusion detection systems. By leveraging the complementarity of these systems, the merged model aims to surpass the individual accuracies, providing a more robust and secure defense against potential threats.

The document outlines the setup instructions required to replicate the project's evaluation process, including the installation of necessary dependencies and the downloading of pre-trained models and datasets. The evaluation script incorporates risk assessment, disaster recovery, and business continuity planning to ensure the project's resilience and uninterrupted operation in the face of security incidents or disasters.

The evaluation process involves training the callback model with early stopping to record accuracy history during training and determine the best weight combination for merging the models. Additionally, the document provides insights into the evaluation results, including accuracy metrics, classification reports, and accuracy history plots.

The integration of risk assessment offers a clear understanding of potential risks associated with the project, helping prioritize risk mitigation strategies to safeguard sensitive data and system integrity. Furthermore, disaster recovery and business continuity strategies ensure the project's ability to recover swiftly and maintain essential operations even in the face of disruptive events.

By combining advanced machine learning techniques, risk management practices, and contingency planning, the "Merging Two Intrusion Detection Systems" project serves as a valuable contribution to the field of cybersecurity, offering a robust defense mechanism against ever-evolving intrusion attempts. Its potential impact extends beyond the academic realm, providing practitioners and organizations with a reliable framework for enhancing their cybersecurity posture and safeguarding their digital assets.

Table of Contents

INTRODUCTION	5
CURRENT SYSTEM	5
PURPOSE OF NEW SYSTEM	5
USER STORIES	
IMPLEMENTED USER STORIES	7
PENDING USER STORIES	10
PROJECT PLAN	
HARDWARE AND SOFTWARE RESOURCES	12
SPRINTS PLAN	13
<i>Sprint 1</i>	13
<i>Sprint 2</i>	13
<i>Sprint 3</i>	14
<i>Sprint 4</i>	15
<i>Sprint 5</i>	16
<i>Sprint 6</i>	17
<i>Sprint 7</i>	18
SYSTEM DESIGN	
ARCHITECTURAL PATTERNS	20

SYSTEM AND SUBSYSTEM DECOMPOSITION	21
DEPLOYMENT DIAGRAM	22
DESIGN PATTERNS	22
...	22
SYSTEM VALIDATION	
.....	23
GLOSSARY	
.....	37
APPENDIX	
.....	38
APPENDIX A - UML DIAGRAMS	38
<i>Static UML Diagrams</i>	38
<i>Dynamic UML Diagrams</i>	40
APPENDIX B - SPRINT REVIEW REPORTS	52
APPENDIX C - USER MANUALS, INSTALLATION/MAINTENANCE DOCUMENT, SHORTCOMINGS/WISHLIST DOCUMENT AND OTHER DOCUMENTS	69
REFERENCES	
.....	80

INTRODUCTION

In this document, we present an introduction to the project "Merging Two Intrusion Detection Systems." The project aims to enhance the accuracy and reliability of intrusion detection mechanisms by combining the strengths of two existing intrusion detection systems. With cybersecurity threats becoming increasingly sophisticated, a robust and efficient intrusion

detection system is crucial to safeguard sensitive data and protect the integrity of systems and networks.

Current System

Currently, intrusion detection systems play a vital role in identifying and mitigating potential security breaches. However, relying solely on a single intrusion detection system may have limitations, and achieving high accuracy can be challenging due to the diversity of attack vectors and evasion techniques. Thus, the need arises to explore novel approaches to enhance the efficacy of intrusion detection.

Purpose of New System

The purpose of the "Merging Two Intrusion Detection Systems" project is to develop a more powerful and effective intrusion detection model by merging two distinct intrusion detection systems. By leveraging the strengths and unique capabilities of each system, we aim to create a more robust and accurate defense against intrusion attempts.

The objectives of the new system are as follows:

Improved Detection Accuracy: By combining the outputs of two intrusion detection systems, we seek to achieve a higher level of accuracy in identifying and classifying potential intrusion events.

Enhanced Resilience: The merged intrusion detection model is expected to exhibit greater resilience to adversarial attacks and evasion techniques, providing a more reliable defense mechanism.

Comprehensive Threat Detection: The new system will be capable of detecting a broader range of cyber threats, including known and emerging attack patterns.

Versatility and Adaptability: The merged model is designed to be adaptable to evolving cybersecurity landscapes, ensuring its effectiveness against emerging threats.

By developing and implementing the "Merging Two Intrusion Detection Systems," we aim to contribute to the field of cybersecurity and offer a practical solution for organizations and individuals seeking to bolster their defense against cyber threats.

USER STORIES

The following section provides the detailed user stories that were implemented in this iteration of the project. These user stories served as the basis for the implementation of the project's features. This section also shows the user stories that are to be considered for future development.

Implemented User Stories

Josue Camejo - Will spend the entirety of the sprints working on researching Intrusion detection systems and how merging IDS systems can prove beneficial to the accuracy percentage. Josue is also working with Christopher as a second Product owner keeping everyone on track and making sure work is being done.

Christopher Puig - Will spend the entirety of the sprints working on researching Intrusion detection systems and how merging IDS systems can prove beneficial to the accuracy percentage. Christopher is the main team lead and will work on keeping the team on track and documenting every member's progress on their sections. He is also in charge of working with the programmers on libraries, software implementation, and how we will be merging the different IDS systems.

Josue Naranjo - In charge of researching and programming the software solution. He will spend most of his time with other team members figuring out what language, libraries, techniques, and algorithms to use for the project. As well as, finalizing all their programmatic implementations into viable data and visualizing that data properly.

Dennis Martinez - In charge of researching and programming the software solution. He will spend most of his time with other team members figuring out what language, libraries,

techniques, and algorithms to use for the project. As well as, finalizing all their programmatic implementations into viable data and visualizing that data properly.

PROJECT PLAN

- 1) Integration of open-source IDSs:
 - a) Investigate the architecture and compatibility of the three IDSs to identify areas for integration.
 - b) Develop a framework or API that facilitates communication and data sharing between the IDSs.
 - c) Implement mechanisms to trigger updates in the other IDSs when one IDS identifies malicious activity.
- 2) Testing, Evaluation, and Optimization:
 - a) Set up a test environment to evaluate the performance and effectiveness of the merged IDS system.
 - b) Conduct integration testing to verify the functionality of the merged IDS and real-time threat intelligence integration.
 - c) Evaluate the impact of the integration on detection capabilities, including coverage of different types of threats and the ability to detect new and emerging threats.
 - d) Fine-tune the integration mechanisms based on evaluation results to optimize performance and effectiveness.
- 3) Documentation and Reporting:
 - a) Document the entire integration process, including design choices, implementation details, and integration configurations.
 - b) Prepare a comprehensive report summarizing the project, highlighting improvements achieved, and providing recommendations for further enhancements.
 - c) Provide clear instructions and guidelines for maintaining and updating the merged IDS system.

Hardware and Software Resources

Hardware Resources:

1. Computer: A high-performance computer with a multi-core processor AMD Ryzen 7 5800X3D and 32GBs of RAM for efficient data processing and model training.
2. GPU: A dedicated GPU, an NVIDIA GeForce RTX 3060 TI, to accelerate deep learning model training and improve overall performance.
3. Storage: Sufficient storage space, an NVME m.2 solid-state drive (SSD), to accommodate the datasets, model checkpoints, and project files.
4. Cloud Services: Microsoft Azure Virtual Machines for scalability and access to more powerful resources if needed.

Software Resources:

1. Python: Python 3.x as the primary programming language for the project.
2. Machine Learning Libraries: Scikit-learn for implementing traditional machine learning algorithms, XGBoost or LightGBM for gradient boosting, and CatBoost for categorical variable handling.
3. Deep Learning Frameworks: TensorFlow & PyTorch for building and training neural network models.
4. Data Preprocessing Libraries: Pandas for data manipulation and cleaning, NumPy for numerical operations on arrays.
5. Visualization Libraries: Matplotlib and Seaborn for data visualization and result plotting.
6. Hyperparameter Tuning Libraries: Scikit-learn's GridSearchCV & RandomizedSearchCV for hyperparameter optimization.
7. IDE (Integrated Development Environment): Visual Studio Code for coding, debugging, and project management.
8. Version Control: Git for version control to track code changes and collaborate with team members.
9. Document Preparation: fMarkdown for creating professional project reports and documentation.
10. Virtual Environments: Virtualenv or conda for managing project dependencies and ensuring a clean development environment.

11. Operating System: The project was developed on various operating systems, such as Windows or Linux, depending on the tests we were running.

Sprints Plan

Sprint 1: System Evaluation and Compatibility Assessment

1. Revising the project proposal.
2. Researching about IDS and datasets.
3. Comparing datasets and IDS accuracy results.

Sprint 2: Integration Framework Development

1. Experimenting with IDS to collect real data collection.
2. Determine the feasibility of merging the three IDS systems.
3. Identify potential integration points and challenges.
4. Evaluate the architecture and compatibility of Snort, Suricata, and Zeek.

Sprint 3: Rule Synchronization Mechanism

1. Develop mechanisms to synchronize rule updates and signatures among the merged IDS systems.
2. Design and develop a framework for communication and data sharing between the IDS systems.
3. Ensure compatibility of data formats and protocols.
4. Implement necessary APIs or connectors to enable seamless integration.
5. Implement an automated process for rule distribution and update propagation.

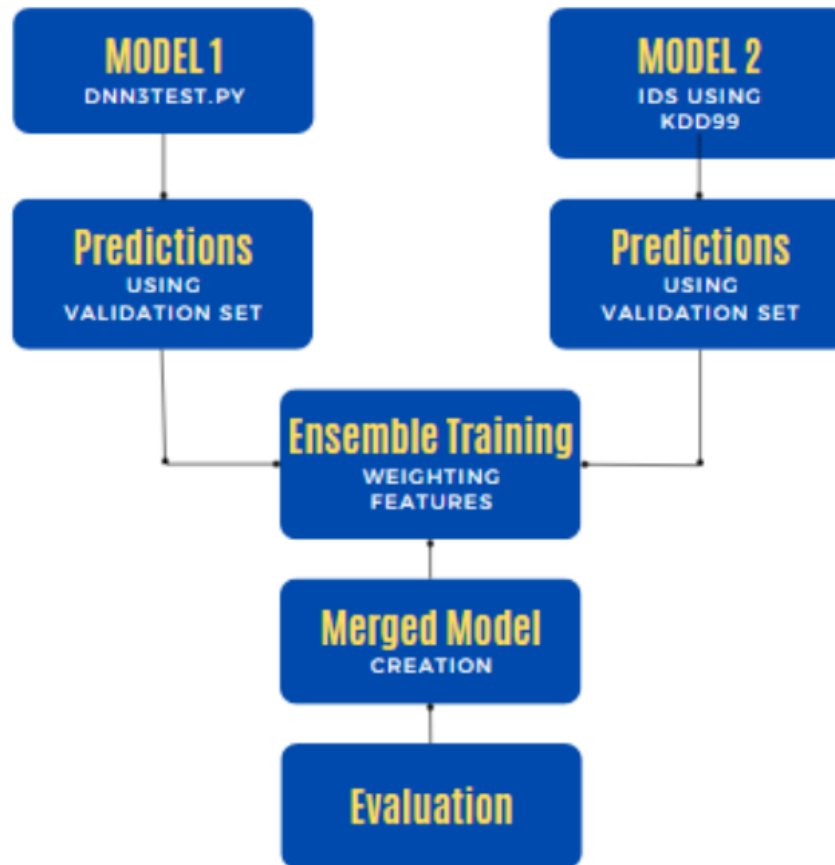
Sprint 4: Performance Testing and Optimization

1. Set up a test environment to evaluate the performance and effectiveness of the merged IDS solution.
2. Test and validate the synchronization mechanism to ensure accuracy and effectiveness.
3. Conduct comprehensive testing to measure detection rates, false positives, and system efficiency.
4. Optimize the system based on performance evaluation results, fine-tuning configurations for optimal results.

Sprint 5: Documentation and Reporting

1. Document the entire integration process, including design choices, implementation details, and configuration settings.
2. Prepare a comprehensive report summarizing the project, including objectives, methodology, results, and recommendations.
3. Provide clear instructions for maintaining and managing the merged IDS solution.

SYSTEM DESIGN



Our systems design is simple yet effective. It begins with the introduction of two Intrusion Detection Systems using their algorithms to predict malicious signatures. To improve the accuracy and precision of the Project's updated IDS system, we begin training the IDS's and merge them together to produce optimal results. In the end, we end up evaluating the results and come to conclusions afterwards.

Architectural Patterns

Importing Libraries:

The code begins by importing necessary libraries, including pandas, numpy, and various components from Keras and scikit-learn. These libraries will be used for data manipulation, model creation, and evaluation.

Data Loading and Preprocessing:

The code reads the training and testing data from two CSV files using pandas. It then performs normalization using the Normalizer class from scikit-learn to scale the features to have unit norm. The data is split into input features (X) and target labels (Y) for training, and similar arrays are created for testing.

Model Definition:

The neural network model is defined using the Sequential class from Keras. It consists of three layers:

The first layer is a Dense layer with 1 neuron and the ReLU activation function.

A dropout layer with a dropout rate of 0.01 is added after the first layer to prevent overfitting.

The second layer is a Dense layer with 64 neurons and the ReLU activation function.

Another dropout layer with a dropout rate of 0.01 is added after the second layer.

Finally, the third layer is a Dense layer with 1 neuron and the sigmoid activation function, suitable for binary classification.

Model Compilation:

The model is compiled with the binary cross-entropy loss function (appropriate for binary classification), the Adam optimizer, and accuracy as the evaluation metric.

Model Training:

The model is trained using the training data (X_train and y_train) with a batch size of 64. During training, two callbacks are used: ModelCheckpoint and CSVLogger. The ModelCheckpoint callback saves the model at the end of each epoch if the loss has improved, and the CSVLogger callback logs the training progress to a CSV file.

Model Evaluation:

After training, the model is evaluated using the test data (X_test and y_test). The predictions are thresholded at 0.5 to obtain binary predictions, and several evaluation metrics are computed, including accuracy, recall, precision, and F1-score. The results are then printed.

Feature Analysis:

The code also performs feature analysis by extracting the weights of the first layer (dense layer) of the trained neural network. These weights are associated with each input feature in the dataset. The features and their corresponding weights are stored in a DataFrame, which is then sorted in descending order of weights. The sorted DataFrame is printed to display the most influential features in the model.

System and Subsystem Decomposition**System Decomposition:****Data Loading and Preprocessing:**

Loading and preprocessing the data is an essential step in any machine learning system. In this code, data is read from remote URLs, and normalization is applied to the features using the Normalizer from scikit-learn. This can be considered as a preprocessing system responsible for preparing the data for training and testing.

Neural Network Model Definition and Compilation:

The neural network model is defined using the Sequential class from Keras. This model defines the architecture and flow of the neural network. The model compilation involves specifying the loss function, optimizer, and evaluation metric. This system is responsible for creating the neural network model and configuring it for training.

Model Training and Evaluation:

The model is trained using the training data and evaluated using the testing data. During training, callbacks are used to save the best model and log the training progress. The evaluation process calculates various metrics like accuracy, recall, precision, and F1-score to assess the model's performance. This system is responsible for training and evaluating the neural network model.

Feature Analysis:

After training, the code performs feature analysis to extract and sort the weights of the first dense layer of the trained neural network. This analysis helps identify the most influential features in the model's decision-making process. This system is responsible for feature analysis and interpretation.

Subsystem Decomposition:

Since the code provided does not have explicit functions or class definitions, the subsystem decomposition is not as evident as in a modular software design. However, we can identify functional blocks or groups of related tasks:

Data Preprocessing Subsystem:

Responsibilities: Reading and preprocessing the data from remote URLs, splitting it into input features and target labels, and applying normalization.

Components: Data loading, data splitting, normalization.

Neural Network Model Subsystem:

Responsibilities: Defining the architecture of the neural network model, compiling it with a loss function and optimizer, and configuring evaluation metrics.

Components: Model architecture definition, model compilation.

Model Training and Evaluation Subsystem:

Responsibilities: Training the model using the training data, evaluating its performance using the testing data, and computing evaluation metrics.

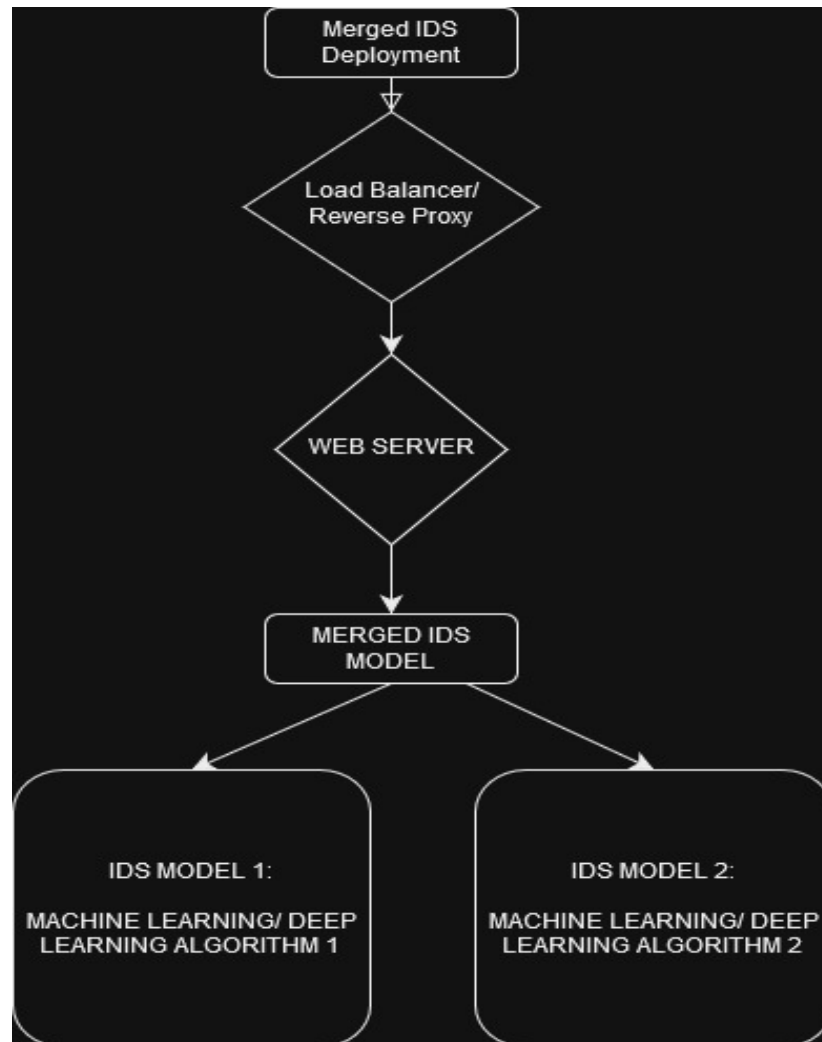
Components: Model training, model evaluation, metric calculations.

Feature Analysis Subsystem:

Responsibilities: Extracting and analyzing the weights of the first dense layer of the trained neural network, sorting and presenting the results.

Components: Weight extraction, feature ranking and presentation.

Deployment Diagram



Design Patterns

-The code implements a binary classification task using a simple feedforward neural network.

- It uses the Keras library with the TensorFlow backend for building and training the neural network.
- The code follows the sequential neural network design pattern, where layers are stacked sequentially to form the model.
- Data preprocessing is applied to the input features using the Normalizer class from scikit-learn.
- Callbacks are utilized during the training process to save the best model and log training progress. This demonstrates the callback design pattern.
- The code calculates various evaluation metrics like accuracy, recall, precision, and F1-score to assess the model's performance, following the metrics and evaluation design pattern.
- Feature analysis is performed by extracting and sorting the weights of the first dense layer of the trained neural network to identify influential features.

SYSTEM VALIDATION

Data Splitting: Before training the model, the dataset is divided into a training set and a testing set. The training set is used to train the model, while the testing set is used to evaluate its performance.

Model Training: The model is trained on the training data using the specified loss function, optimizer, and evaluation metric. During training, the model is updated to minimize the training loss.

Model Evaluation: After training, the model is evaluated on the testing set to assess its performance on unseen data. Evaluation metrics such as accuracy, recall, precision, and F1-score are calculated to measure the model's effectiveness.

Hyperparameter Tuning: To optimize the model's performance, hyperparameters (e.g., learning rate, batch size, number of neurons) can be tuned through techniques like grid search or random search.

Feature Analysis Validation: The feature analysis step involves extracting and sorting the weights of the first dense layer of the trained neural network. To validate the feature analysis, domain experts or further statistical analyses may be used to confirm the significance of the identified influential features.

Cross-Validation (Optional): To further ensure the model's generalization performance, cross-validation can be used. It involves partitioning the data into multiple subsets (folds) and training/evaluating the model on different combinations of these subsets.

GLOSSARY

Pandas: A popular Python library for data manipulation and analysis, offering data structures like DataFrame and Series.

NumPy: A fundamental Python library for numerical computing, providing support for arrays and mathematical operations.

Keras: An open-source deep learning API written in Python, designed for easy and fast prototyping of neural networks.

TensorFlow: An open-source deep learning framework developed by Google, widely used for building and training machine learning models.

Sequential: A Keras class representing a linear stack of neural network layers, suitable for simple feedforward models.

Dense Layer: A fully connected neural network layer, where each neuron in the layer is connected to every neuron in the previous and subsequent layers.

Dropout: A regularization technique used to prevent overfitting in neural networks by randomly setting a fraction of input units to 0 during training.

Activation Function: A function that introduces non-linearity into the neural network, allowing it to learn complex patterns.

Binary Cross-Entropy: A loss function used in binary classification tasks to measure the difference between predicted and actual class probabilities.

Optimizer: An algorithm used to adjust the weights of a neural network during training to minimize the loss function.

Metrics: Performance measures used to evaluate the model's accuracy and effectiveness, such as accuracy, precision, recall, and F1-score.

Normalizer: A scikit-learn class used to scale input features to have unit norm, often employed in preprocessing.

Model Checkpoint: A Keras callback that saves the best model during training based on specified criteria like loss improvement.

CSVLogger: A Keras callback that logs training metrics to a CSV file during training.

Feature Analysis: The process of analyzing the importance of individual features in the model's decision-making process.

Batch Size: The number of samples used in each training iteration to update the model's weights.

Epoch: One complete pass of the training data through the neural network during training.

Binary Classification: A classification task where the model assigns each input to one of two possible classes.

Accuracy: The proportion of correctly predicted instances over the total number of instances in the evaluation set.

Recall: The ability of the model to correctly identify positive instances out of all actual positive instances.

Precision: The ability of the model to correctly identify positive instances among the instances predicted as positive.

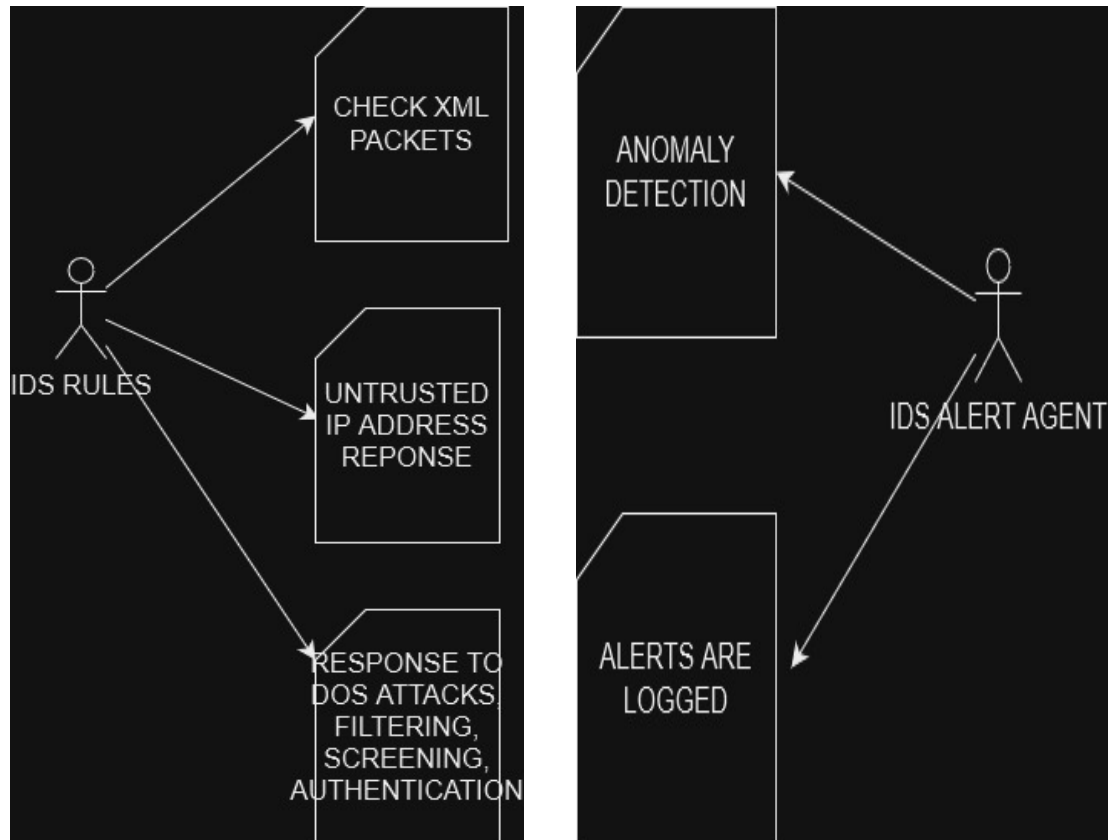
F1-Score: The harmonic mean of precision and recall, providing a balance between the two metrics.

Weights: Parameters in a neural network that are learned during training to make predictions.

Neural Network: A computational model inspired by the human brain's architecture, used for machine learning and pattern recognition tasks.

Training Data: The dataset used to train the neural network model.

Testing Data: The dataset used to evaluate the neural network model's performance.

APPENDIX**Appendix A - UML Diagrams**

Appendix B - Sprint Review Reports

Sprint Review 1 : During the sprint we were able to properly discuss what we learned throughout the first sprint and get out information and schedules organized and setting up a project proposal for the product owner.

Sprint Review 2 : During the sprint we were able to properly discuss what we learned throughout the first and second sprint and get out information and schedules organized and still revise our project proposal for the product owner to review to accept an idea for the future sprints.

Sprint Review 3 : During the sprint we were able to properly discuss what we learned throughout the third sprint and get out information and schedules organized and do more research on the project proposal that has been accepted by the product owner.

Sprint Review 4 : During the sprint we were able to properly discuss what we learned throughout the third sprint and get out information and schedules organized. We are currently working on merging IDS data to receive a greater result

Sprint Review 5 : During the sprint we are trying to provide more graphs to showcase the models we have created and also trying to finish the list of deliverables and also the videos required for this project.

Appendix C - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents

Appendix C

<https://docs.google.com/document/d/1JtDEEHAWHWAJoNJNCALNEXM7cDkHcARvA2W91BkPPAo/edit?usp=sharing>

REFERENCES

- Antonio Sanchez, Joan Melià-Seguí, Jordi Herrera-Joancomartí, "**Combining Snort and Suricata for Improved Network Intrusion Detection**", edited by Antonio Sanchez, Sanchez, 2016
- Victor A. A. Ramos, Andre L. M. dos Santos, Pedro R. M. Inácio, Jorge M. Nogueira, Title: "**A Distributed and Cooperative Intrusion Detection System Based on Zeek**", edited by Victor A, 2020
- Roesch, Martin. **Snort - Lightweight Intrusion Detection for Networks**. 12 Nov. 1999, pp. 229–238. Accessed 9 June 2023.
- Ferriyan, Andrey, et al. "**Generating Network Intrusion Detection Dataset Based on Real and Encrypted Synthetic Attack Traffic.**" *Applied Sciences*, vol. 11, no. 17, 26 Aug. 2021, p. 7868, <https://doi.org/10.3390/app11177868>. Accessed 20 Nov. 2021.

Model 2

<https://www.kaggle.com/code/wailinnoo/intrusion-detection-system-using-kdd99-dataset>