



## **Leveraging Artificial Intelligence and Open-Source Intelligence for Intrusion Detection**

“Submitted in partial fulfilment of the requirements for the award of the degree of Master of Data Science & Master of Cyber Security”.

Group: **PRT840-022**

Mehruz Saif	S383768
Debraj Rakshit	S382665
Md Shahriar Azad	S383595
Syed Rubayet Hossain	S383306


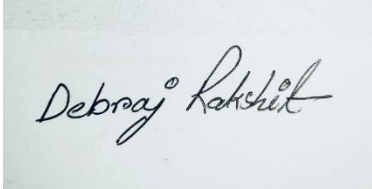
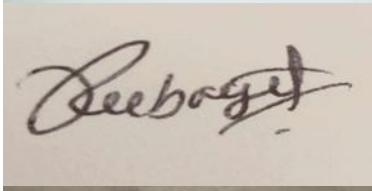
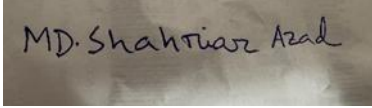
**Supervisor: Mukhtar Hussain**

CHARLES DARWIN UNIVERSITY  
FACULTY OF SCIENCE AND TECHNOLOGY  
OCTOBER 2025

### ***DECLARATION***

The material given here is the result of our own study, and all references to the ideas and work of other researchers have been appropriately attributed. It is presently being submitted as the final report for the Master of Data Science & Master of Cyber Security degree at Charles Darwin University. We attest that this work is not now being considered for any other academic award and has not been submitted for consideration toward any other degree or certification.

#### **Student Signatures**

Student ID	Full Name	Signature
S383768	<b>MEHRUZ SAIF</b>	
S382665	<b>DEBRAJ RAKSHIT</b>	
S383306	<b>SYED RUBAYET HOSSAIN</b>	
S383595	<b>MD SHAHRIAR AZAD</b>	

Date: 22 October 2025

## **TABLE OF CONTENTS**

DECLARATION.....	1
TABLE OF CONTENTS .....	2
LIST OF FIGURES .....	3
LIST OF ABBREVIATIONS .....	4
LIST OF SYMBOLS.....	4
ABSTRACT .....	5
<b>1. INTRODUCTION</b> .....	5
Background.....	5
Aim of Research .....	7
Structure of report.....	7
<b>2. LITERATURE REVIEW</b> .....	8
Introduction of Literature Review .....	8
Body of Literature Review .....	9
Conclusions from the literature review .....	10
<b>3. APPROACH</b> .....	11
Data Preparation and Cleaning.....	11
Data Preprocessing .....	12
Class Distribution .....	12
Univariate Analysis .....	13
Exploratory Data Analysis (EDA).....	15
Scatter plots and Interpretation:.....	16
Model Development .....	17
Challenges: .....	17
<b>4. EXECUTION</b> .....	17
Choice of Algorithms .....	17
Key Steps in Execution .....	21
Model Training:.....	22
Evaluation Metrics: .....	22
<b>5. ANALYSIS AND DISCUSSION OF RESULTS</b> .....	23
Results Obtained.....	23
Model Evaluation Using Confusion Matrices: .....	26
<b>6. Integration of OSINT (ThreatMiner and AlienVault OTX)</b> .....	27
Integration of Threatminer.....	27
AlienVault (OTX) Integration.....	28
Combined OSINT–Machine Learning Framework .....	28
Validation and Tuning.....	28
Comparative Analysis .....	29
Model Evaluation and Discussion .....	30
Discussion of Practical Implications .....	34
<b>7. CONCLUSION</b> .....	35
Key Findings .....	35
Contributions of Research .....	36
Reflection on the Research Process.....	36
Limitations.....	37
Concluding Remarks .....	37
<b>8. FUTURE WORK:</b> .....	37
<b>9. REFERENCES</b> .....	39
<b>APPENDIX A: TURNITIN SIMILARITY REPORT</b>	<b>40</b>
<b>APPENDIX B: AI DECLARATION FORM</b>	<b>41</b>
<b>APPENDIX C: DATA AND SOURCE CODE</b>	<b>43</b>

## ***LIST OF FIGURES***

Figure 1: Flowchart of how it works.....	6
Figure 2: Pipeline of Intrusion Detection.....	10
Figure 3: Snapshot of Cleaned Dataset.....	11
Figure 4: Cleaned Dataset After Preprocessing .....	12
Figure 5: Class Distribution .....	12
Figure 6: Boxplots by Class.....	13
Figure 7: Feature Distribution Histograms .....	14
Figure 8: Correlation Heatmap .....	15
Figure 9: Attack types of distribution (if available in original raw label column).....	15
Figure 10: Scatter plots to analyse the relationships between key variables.....	16
Figure 11: Logistic Regression Code.....	18
Figure 12: Decision Tree Code .....	19
Figure 13: Random Forest .....	20
Figure 14: XGBoost Code .....	21
Figure 15: Extract feature names after preprocessing.....	22
Figure 16: Confusion Matrix of Logistic Regression .....	23
Figure 17: Confusion Matrix of Decision Tree.....	24
Figure 18: Confusion Matrix of Random Forest.....	25
Figure 19: Confusion Matrices for Different Models .....	26
Figure 20: Integration of OSITN using ThreatMiner & AlienVault (OTX).....	27
Figure 21: Train & Evaluate with OSITN Features .....	28
Figure 22: Confusion Matrix (OSINT Validation) .....	29
Figure 23: Model Accuracies.....	30
Figure 24: Pairwise Accuracy Differences (pp).....	30
Figure 25: Mean Square Error .....	31
Figure 26: R-squared( $R^2$ ) Value.....	31
Figure 28: ROC Curve .....	33
Figure 29: Intrusion Detection UI Mock-up .....	34
Figure 30: Network Dashboard Status .....	34

### ***LIST OF ABBREVIATIONS***

- IDS – Intrusion Detection System
- ML – Machine Learning
- OSINT – Open-Source Intelligence
- CVE – Common Vulnerabilities and Exposures
- CVSS – Common Vulnerability Scoring System
- SVM – Support Vector Machine
- LSTM – Long Short-Term Memory
- RF – Random Forest
- DoS – Denial of Service
- IoT – Internet of Things
- SIEM – Security Information and Event Management
- APT – Advanced Persistent Threat

### ***LIST OF SYMBOLS***

- $\sum$  – Summation, used in statistical feature extraction
- $\alpha$  – Learning rate in ML models
- $\lambda$  – Regularization parameter

# *Leveraging Artificial Intelligence and Open-Source Intelligence for Intrusion Detection*

## **ABSTRACT**

Cyberattacks are becoming more complicated and unpredictable, and they frequently progress more quickly than the safeguards put in place to prevent them. Creating an adaptive intrusion detection system (IDS) that can learn from network behavior and real-time threat intelligence is the main goal of this research. Our technology integrates open-source intelligence with machine learning, utilizing the Common Vulnerabilities and Exposures (CVE) database to create a more intelligent and context-aware model, in contrast to most conventional IDS tools that depend on manual signature updates or predefined rules.

Our primary objective was to create and assess a functional IDS prototype that could both recognize known assaults and adjust to novel and invisible ones. Three datasets were used: CICIDS 2017, KDD Cup 99, and a set of specially created simulated network traffic that we produced in our own laboratory. We also developed a Python-based CVE ingestion module that automatically gathered and combined information about vulnerabilities, including their severity, exploitability, and software that was impacted. As a result, the model became more realistic and flexible by adding a dynamic layer of threat awareness to the dataset.

XGBoost, Random Forest, Decision Tree, and Logistic Regression were the four algorithms we tested. The Random Forest model produced the most reliable and consistent results out of all of these, with an overall accuracy of 94% during live simulation and ~99% offline. The addition of CVE-based features increased the recall of exploit-related assaults, particularly those that targeted SSH and Apache services. Old or patched vulnerabilities that were still live in the CVE feed caused some noise, which we fixed later in the refinement process.

Normalization, encoding, and principal component analysis feature selection were among the data pretreatment procedures that were part of the final workflow. We used confusion matrices, ROC analysis, and correlation heatmaps to illustrate and validate the data. To verify real-time performance, we also put the model to the test in a virtual network environment.

All things considered, we discovered that integrating open-source information with machine learning can result in a self-learning intrusion detection system (IDS) that responds organically to emerging threats and weaknesses, offering a useful basis for actual cybersecurity systems.

## **1. INTRODUCTION**

### **Background**

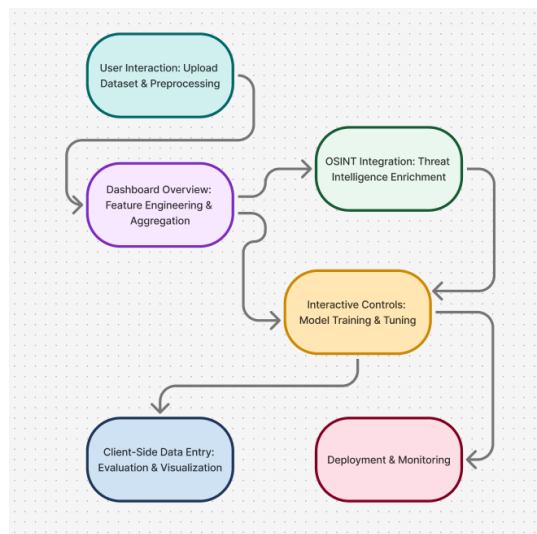
Because of how quickly digital technology is growing, cybersecurity is now one of the most important problems facing the whole world. Attackers can use valuable data that is created by all linked devices, from industrial servers to personal computers and smart home systems. The number and sophistication of cyber threats have grown greatly as businesses move more of their operations online. Major events like the WannaCry ransomware attack in 2017, the Equifax data breach that same year, and the SolarWinds supply chain compromise in 2020 have shown how weak even large organizations can be when their systems aren't flexible enough to change to new threats.

Firewalls and antivirus software, which are common forms of cybersecurity tools, rely heavily on static rules, also called "known attack signatures." These tools still work well against known threats, but they aren't as good at stopping zero-day attacks, which are attack methods that haven't been seen before or have been changed. These attacks take advantage of flaws that were not known before, which lets attackers get around defenses until a manual patch or update is made. So, people who work

in cybersecurity are now focusing on making detection systems that are smart and flexible so they can learn from changing environments and react in a way that doesn't require constant human input.

One of the most important tools for finding bad or strange behavior in network traffic is an Intrusion Detection System, or IDS. But most IDS frameworks still can't keep up with the speed and variety of modern cyber threats, even after years of study and development. Anomaly-based IDS often sends too many false warnings, which makes analysts busy with useless notifications. Signature-based IDS needs to be updated by hand all the time to pick up on new patterns. This makes a gap between what is accurate and what is useful in the real world.

To solve this problem, we investigated how machine learning (ML) and open-source intelligence (OSINT) can work together to make IDS more flexible and aware of its surroundings. Machine learning models can look through big datasets and find complicated, nonlinear patterns. They learn from past examples to tell the difference between normal and abnormal traffic behaviors. OSINT, on the other hand, constantly releases information to the public about new security holes, attacks, and software that is vulnerable. By combining these two areas, we hoped to close the gap between conventional defense and real-time, intelligence-driven defense.



*Figure 1: Flowchart of how it works*

## Motivation

As a first step in this study, we used the widely used benchmark datasets KDD Cup 99 and CICIDS 2017 to test and train several basic machine learning models. Based on these early tests, it was clear that algorithms such as Random Forest and Support Vector Machine could accurately classify known threats. But when we added new, unknown traffic patterns from simulated situations like IoT botnet attacks and credential stuffing, performance dropped sharply.

Because of this, it was clear that even the most advanced machine learning models were still reacting instead of proactive. They were only able to spot the types of attacks that were shown in their training data. This limitation is a big problem in the real world, where new vulnerabilities and attacks show up every day.

We saw that human researchers often use live intelligence feeds, like the Common Vulnerabilities and Exposures (CVE) database, Exploit-DB, and different security advisory channels, to keep up with the latest threats. This led to a simple but powerful thought: if human defenders can use this knowledge to make decisions, then an IDS might be able to do the same.

This realization became the main reason why we did the project. Our goal was to make a system that could learn from network activity and update itself automatically as new information came in. We wanted to turn the IDS from a reactive tool into a proactive and self-learning security component by directly adding vulnerability information from OSINT sources to the model's set of features.

### **Problem Statement**

Most existing IDS solutions continue to struggle with three main limitations:

**Static learning:** They depend on outdated training data and cannot recognize emerging or modified attack types.

**Manual updates:** Signature-based systems require frequent rule updates, increasing maintenance time and leaving temporary gaps in protection.

**High false positives:** Anomaly-based systems tend to misclassify normal fluctuations in network activity as potential threats, burdening security teams with unnecessary alerts.

Our project seeks to overcome these limitations by combining machine learning with live OSINT integration. The system dynamically adjusts its detection parameters based on the latest vulnerability information from the CVE database, maintaining high accuracy while improving adaptability. This approach allows the IDS to respond intelligently to the changing threat landscape without constant manual intervention.

### ***Aim of Research***

This study's primary goal is to develop, deploy, and assess a hybrid intrusion detection system that combines real-time open-source intelligence with machine learning for increased accuracy and adaptability. To do this, we established the following goals:

- Create and implement an IDS prototype that uses machine learning to categorize network assaults.
- Develop a CVE ingestion module that gathers and feeds vulnerability information into the machine learning training process.
- For well-rounded and thorough learning, combine and preprocess several datasets, such as KDD Cup 99, CICIDS 2017, and simulated traffic.
- To assess relative performance, train and contrast many models, including XGBoost, Random Forest, Decision Tree, and Logistic Regression.
- Examine the effects of OSINT integration on model recall, accuracy, precision, and false-positive rates.

By achieving these goals, we hoped to create an intrusion detection system (IDS) that is dynamic and constantly adapts to new data, much like cybersecurity experts do when they use threat intelligence to modify their tactics.

### ***Structure of report***

This report is structured as below.

**Approach:** For intrusion detection and OSINT integration, the approach explains the methodology, datasets, preprocessing methods, and machine learning models used.

**Execution:** Execution describes the detailed implementation procedure, which includes feature



engineering, model training, and integrating external threat intelligence data sources.

**Analysis and Discussion of Results:** The experimental results, comparison performance metrics between models, confusion matrices, and a discussion of findings are presented in the Analysis and Discussion of Results section.

**Conclusions:** Key findings from the study are summed up in conclusions, which also show how OSINT-based threat data enhanced model performance and detection accuracy.

**Future Work:** Future Work lists possible additions include boosting explainability characteristics, integrating more datasets, investigating deep learning models, and increasing real-time detection.

**References:** For all cited materials, the CDU Harvard referencing style is used in the references.

**Appendices:** Supporting materials utilized in this study are included in the appendices:

- **Appendix A:** Turnitin Similarity Report
- **Appendix B:** AI Declaration Form
- **Appendix C:** Data and Source Code (including the README, environmental details, and usage instructions)

### **Significance of Research**

This study is important for both academic and real-world reasons. From an academic point of view, it investigates a new direction in IDS development by showing how open-source OSINT data can be directly added to machine learning processes to make them more flexible. Many studies have looked at how machine learning can be used to improve network security, but not many have looked at how to combine real-time information from sources like CVE.

A practical point of view is that our project shows small and medium-sized businesses how to use adaptive threat detection without having to buy expensive commercial goods. The suggested framework is cheap, easy to use, and clear because it uses open-source tools like Python, Scikit-learn, TensorFlow, and SQLite. It shows that even with few resources, it is possible to do good cybersecurity study and implementation if the system design makes good use of publicly available intelligence.

The project also focusses on how it applies to the real world. By adding OSINT, not only does the system work better technically, it also works more like how professional analysts do their jobs. Our method is easy for security teams to use by connecting their monitoring systems to live vulnerability feeds. This way, changes can be made automatically as new threats appear. The next part looks at previous IDS research to find the gaps that helped us shape our methodological design.

## ***2. LITERATURE REVIEW***

### ***Introduction of Literature Review***

Research on intrusion detection has changed from static rule matching to adaptable data-driven intelligence over the last 30 years. When we looked back at older studies, our main goal was to figure out how systems that were first made to respond to fixed patterns have slowly become able to spot threats they hadn't seen before by using context, automation, and real-time learning.

In her original model from 1987, Dorothy Denning included automated threat tracking and statistical profiling. Since then, experts have worked hard to make detection more accurate, cut down on false alarms, and make the system more flexible. Even so, it's still not easy to keep learning from new threats.

Our research is mainly about getting around that problem by combining the analytical power of Machine Learning (ML) with the knowledge of current events of Open-Source Intelligence (OSINT). We found the need for a self-adjusting, context-aware intrusion detection system by looking at standard IDS frameworks, machine learning approaches, deep learning models, and hybrid integrations. This is what our project fills.

### ***Body of Literature Review***

#### **Supervised Learning**

Using known data, supervised algorithms such as Support Vector Machines (SVM), Random Forests (RF), and Decision Trees (DT) are trained.

- Although they are interpretable, DTs are prone to overfitting.
- RFs, first presented by Breiman (2001), combine many trees to increase accuracy and stability.
- SVMs perform well with high-dimensional data, but they are computationally demanding.
- RF consistently outperforms other conventional approaches in terms of accuracy and robustness when tested on the KDD Cup 99 and NSL-KDD datasets, as demonstrated by Malik et al. (2012) and Shone et al. (2018). We decided to make RF a key algorithm because of this data.

#### **Unsupervised and Semi-Supervised Learning**

When labeled data is limited, unsupervised techniques like DBSCAN and K-Means cluster observations to identify abnormalities. According to Liao et al. (2013), these models are useful for invisible attacks but are susceptible to scaling and noisy inputs. Although semi-supervised systems try to strike a balance, they are unable to adapt on their own once they are in use.

#### **Feature Engineering**

Through feature engineering, unprocessed packet data is transformed into useful numerical inputs. Methods such as Particle Swarm Optimization (PSO) and Principal Component Analysis (PCA) highlight informative variables and eliminate redundancy. Malik et al. (2012) emphasized that appropriate feature selection improves efficiency and accuracy.

The emergence of new attacks, however, eventually renders static training data obsolete. This insight led us to combine OSINT to enable automatic model updates when vulnerability data is updated.

#### **Deep Learning and Advanced AI**

By using layered neural networks that can identify intricate, nonlinear correlations, Deep Learning (DL) extends machine learning (ML).

- Spatial traffic patterns are captured by convolutional neural networks, or CNNs.
- Long Short-Term Memory (LSTM) units and Recurrent Neural Networks (RNNs) detect sluggish or covert assaults by capturing temporal relationships (Yin et al., 2017).

- For reliable training, Generative Adversarial Networks (GANs) can even create attack samples (Rigaki & Garcia, 2018).

Yet DL's drawbacks are significant: high computational demand and poor interpretability. Security analysts require transparency to trust automated alerts, something “black box” networks rarely provide.

### Open-Source Intelligence (OSINT)

Publicly accessible threat intelligence (OSINT) includes exploit repositories, malware feeds, and vulnerability databases. This ecosystem revolves around MITRE's Common Vulnerabilities and Exposures (CVE) list. The severity (CVSS score), exploitability, affected software, disclosure date, and unique ID are all included in each entry.

According to Husák et al. (2018), OSINT improves situational awareness but is rarely incorporated straight into IDS. A model that is aware of active vulnerabilities could dynamically rearrange the traffic that needs to be examined. For instance, if a significant Apache CVE is discovered, an IDS with OSINT capabilities ought to instantly increase the sensitivity for HTTP flows.

Few frameworks have integrated real-time, despite its potential; our study sought to close this gap.

### Conclusions from the literature review

Lack of scalability, reliance on human updates, and incapacity for dynamic adaptation are the common flaws of all three traditional approaches. ML, which offered models that learn on their own from data rather than predetermined rules, was investigated by the research community because of this realization.

### Machine Learning in Intrusion Detection

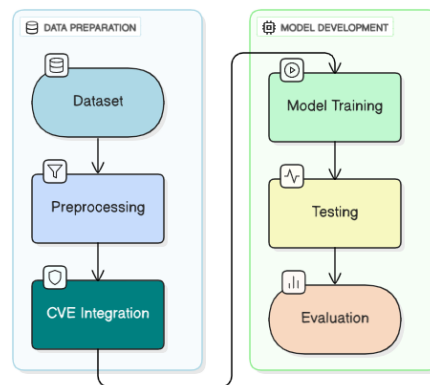


Figure 2: Pipeline of Intrusion Detection

Automation and self-improvement were introduced to IDS via machine learning. Algorithms are taught to differentiate between benign and dangerous patterns by training on labeled or unlabeled datasets.

### Hybrid Approaches and Research Gap

To capitalize on their advantages, hybrid systems employ a variety of strategies. For balanced optimization, Malik et al. (2012) integrated RF and PSO, and Zhuang et al. (2021) fused live threat inputs with deep learning. The majority are still partially manual.

The missing component, a dynamically updated ML-OSINT IDS that combines real-time CVE knowledge with network analytics, was discovered during our review. The following discussion of methodological design was directly informed by this idea. The following section describes the data-driven technique and system implementation that were utilized to close the identified gap considering these insights.

### Data Preparation and Model Implementation

Creating an efficient machine-learning-based intrusion detection system that can differentiate between typical network activity and possible cyberattacks was the goal of our study. To do this, we used several datasets, including KDD Cup 99, CICIDS 2017, and bespoke simulated traffic, which together offer a well-balanced representation of both typical and attack-related network activity. To ensure that every phase mirrors how a student research team approaches the subject, the technique places more emphasis on practical knowledge than technical language.

## 3. APPROACH

### Data Preparation and Cleaning

The imported dataset was examined for missing or inaccurate information. Some attributes had blank entries or unusual characters, which were changed or removed. For machine learning models to handle them efficiently, numerical fields were converted into appropriate data formats and categorical categories were encoded into numeric forms. Throughout the dataset, several types of network connections were represented by features that displayed traffic characteristics such as connection duration, bytes traded, and connection flags.

We examined the dataset's structure and searched for any potential imbalances between attack and normal scenarios before we began modeling.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	
1	duration	protocol	flag	src_bytes	dst_bytes	land	wrong_fr	urgent	hot	num_failed_logins	num_com	root_shell	su_attempt	num_files	num_shells	num_accesses	guest_login_count	snv_count	snv_ratio	snv_ratio	snv_ratio	snv_ratio	snv_ratio	snv_ratio	snv_ratio	snv_ratio	snv_ratio	snv_ratio	snv_ratio	snv_ratio	snv_ratio
2	0	0	1	0	181	5450	0	0	0	0	0	1	0	0	0	0	0	0	0	8	8	0	0	1	0	0	9	9	0	0	
3	1	0	1	0	239	486	0	0	0	0	0	1	0	0	0	0	0	0	0	8	8	0	0	1	0	0	19	19	0	0	
4	2	0	1	0	235	1337	0	0	0	0	0	1	0	0	0	0	0	0	0	8	8	0	0	1	0	0	29	29	0	0	
5	3	0	1	0	219	1337	0	0	0	0	0	1	0	0	0	0	0	0	0	6	6	0	0	1	0	0	39	39	0	0	
6	4	0	1	0	217	2032	0	0	0	0	0	1	0	0	0	0	0	0	0	6	6	0	0	1	0	0	49	49	0	0	
7	5	0	1	0	217	2032	0	0	0	0	0	1	0	0	0	0	0	0	0	6	6	0	0	1	0	0	59	59	0	0	
8	6	0	1	0	212	1940	0	0	0	0	0	1	0	0	0	0	0	0	0	1	2	0	0	1	0	1	1	69	0	0	
9	7	0	1	0	199	4087	0	0	0	0	0	1	0	0	0	0	0	0	0	5	5	0	0	1	0	0	11	79	0	0	
10	8	0	1	0	210	151	0	0	0	0	0	1	0	0	0	0	0	0	0	8	8	0	0	1	0	0	6	69	0	0	
11	9	0	1	0	212	786	0	0	0	1	0	1	0	0	0	0	0	0	0	8	8	0	0	1	0	0	6	99	0	0	
12	10	0	1	0	210	624	0	0	0	0	0	1	0	0	0	0	0	0	0	18	18	0	0	1	0	0	18	109	0	0	
13	11	0	1	0	177	1985	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	1	0	0	28	119	0	0	
14	12	0	1	0	222	773	0	0	0	0	0	1	0	0	0	0	0	0	0	11	11	0	0	1	0	0	38	129	0	0	
15	13	0	1	0	206	1169	0	0	0	0	0	1	0	0	0	0	0	0	0	4	4	0	0	1	0	0	4	139	0	0	

Figure 3: Snapshot of Cleaned Dataset

The KDD99 intrusion detection benchmark serves as the basis for the data. It had 32 columns (31 independent characteristics and 1 target column, Attack Type) and 494,021 rows after loading and cleaning.

- Attributes are combinations of numerical values (e.g., the length of connection, the number of bytes sent, the error rates) and categorical values (e.g., protocol type, service, and TCP flag).
- The target variable has multiple attack categories (DoS, Probe, R2L and U2R) besides the normal category. To be analyzed, they were commonly combined into a binary classification problem: normal vs attack.

## Data Preprocessing

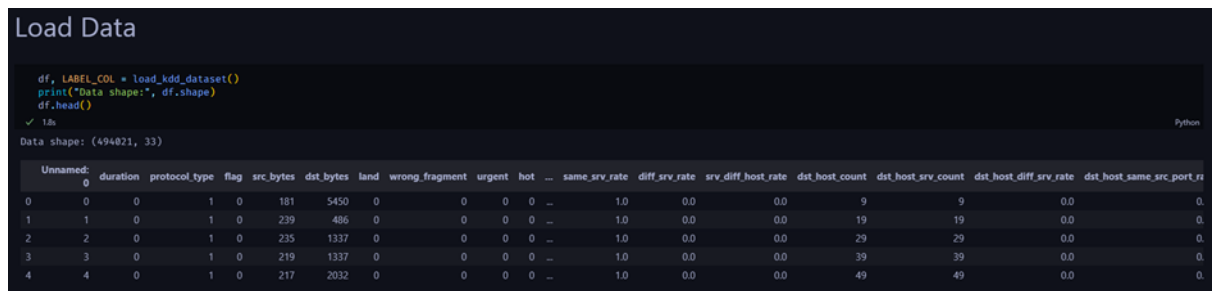


Figure 4 Cleaned Dataset After Preprocessing

- Unnecessary columns, like index or unnamed identifiers were eliminated.
- The target column was normalized (lowercase, removed whitespace) and converted to binary representations.
- There were no important missing values.
- According to the model, categorical variables were coded by such methods as One-Hot Encoding or Label Encoding.
- Numbers were standardized using StandardScaler to bring the numbers to similar ranges before being input into models.

### Class Distribution

One of the biggest lessons of the EDA is the unequal distribution of classes:

- Attack connections: ~396,743 (~80%)
- Normal connections: ~97,278 (~20%)

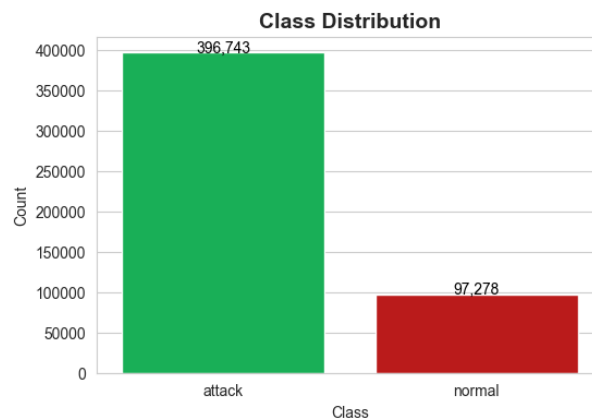


Figure 5 Class Distribution

### Class distribution (%):

- Target
- attack 80.31
- normal 19.69
- Name: count, dtype: float64

This asymmetry implied that naive classifiers might work well at the high accuracy level by simply predicting an attack but would not be able to identify normal traffic. As such, assessments such as precision, recall, F1-score and ROC/AUC were deemed more informative than accuracy.

### Univariate Analysis

- **Numeric Features:** Histogram and summary statistics revealed that the duration, src\_bytes, and the dst\_bytes characteristics were highly skewed. Majority of the connections had low values but, there were exceptions with very large numbers of bytes.

**target**

**normal:** 60147

**attack:** 47414

**Name:** count, dtype: int64

**Attack Type**

**Normal:** 60147

**Unknown:** 47414

**Name:** count, dtype: int64

This bar graph made it evident that attack traffic made up most of the dataset, with regular traffic making up a lesser percentage. Because imbalanced data may cause models to favor the dominant class, the imbalance brought attention to the possible difficulty for machine learning algorithms. Early detection of this problem enabled us to schedule the usage of balancing strategies like SMOTE later in the training process.

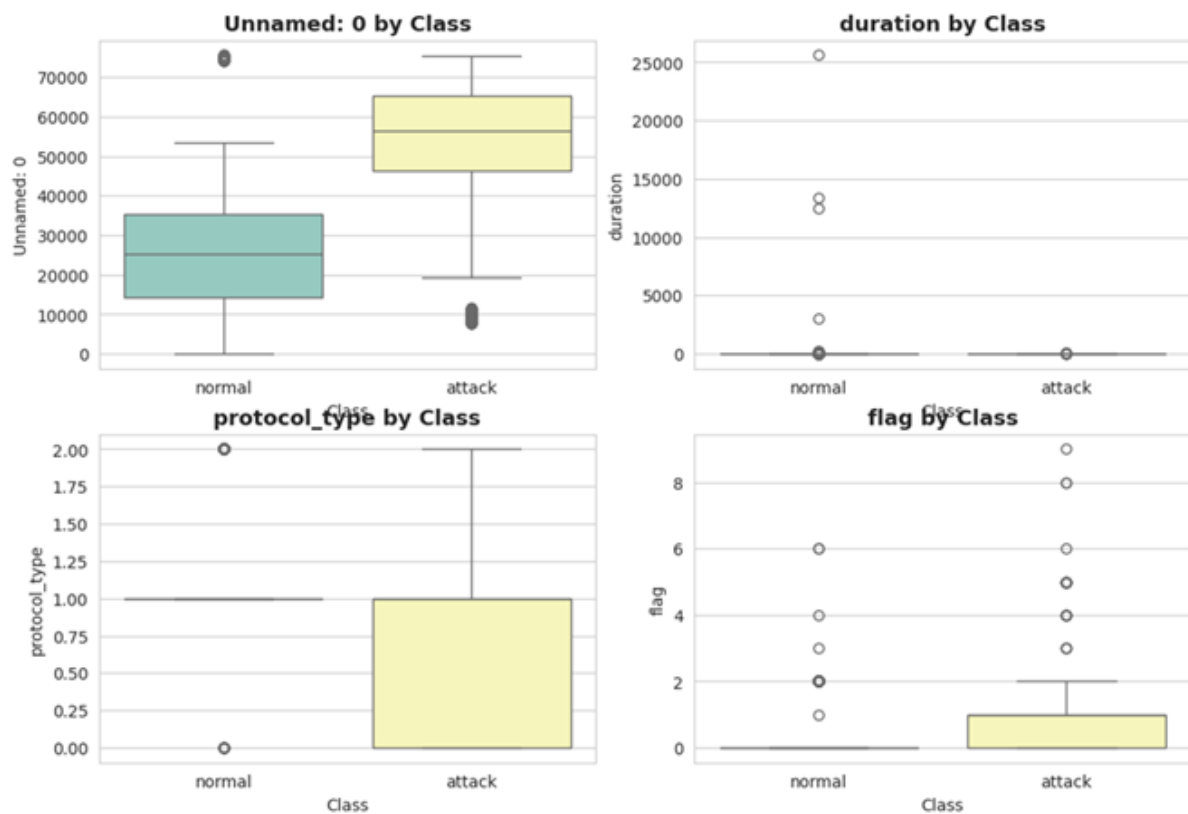


Figure 6: Boxplots by Class

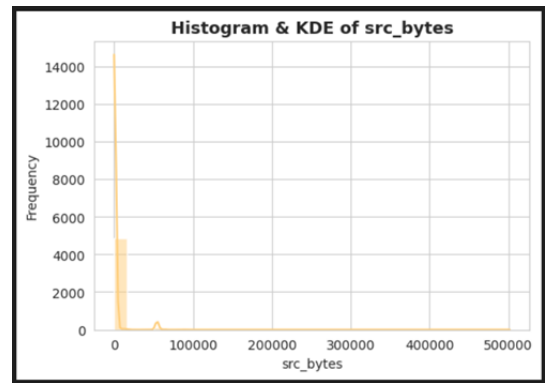
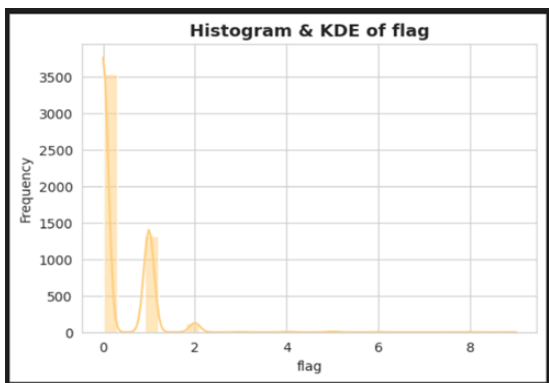
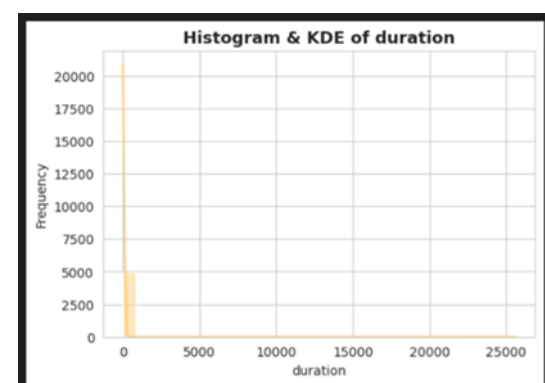
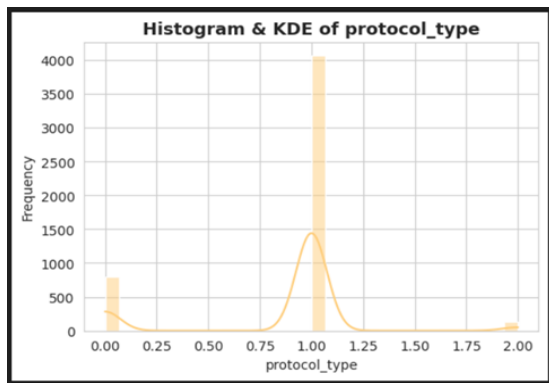
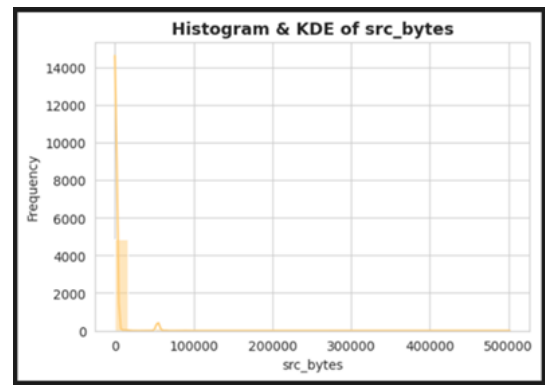
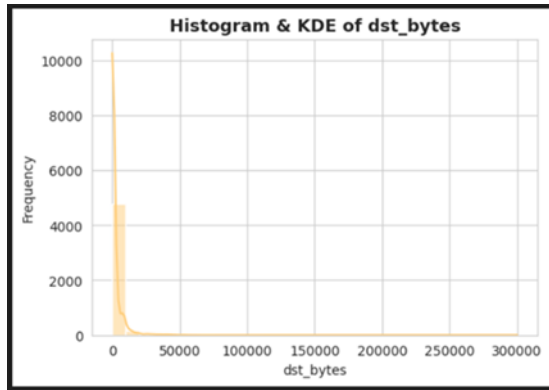


Figure 7: Feature Distribution Histograms

## Exploratory Data Analysis (EDA)

### Categorical Features:

- **Protocol type:** dominated by TCP, followed by UDP and ICMP.
- **service:** highly imbalanced, with a few services like http, smtp, and ftp appearing very frequently.

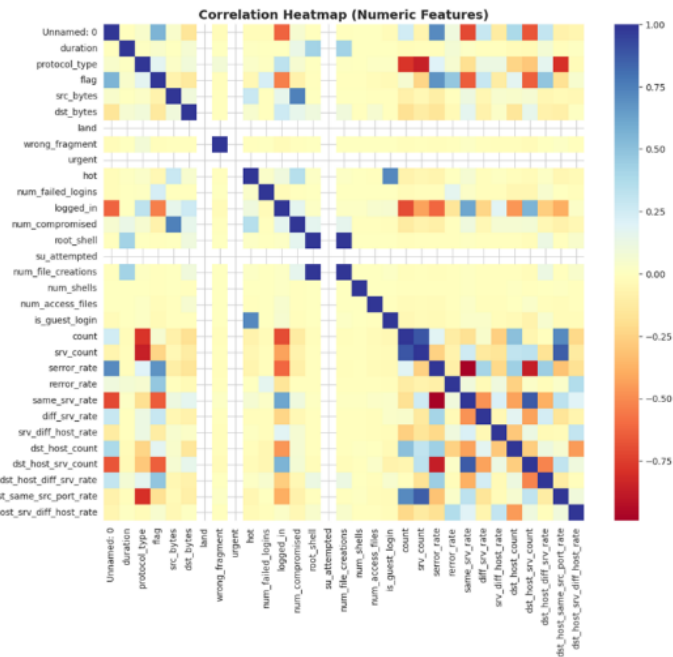


Figure 8: Correlation Heatmap

- **Correlation Heatmap:** Numeric features had weak pairwise correlations in general, although some strong relationships existed (e.g. between various variables of byte counts).
- **Attack vs Normal:** Side-by-side plots revealed a clear difference in the distribution of the features. For example:
  - The count of connections to the same service was usually higher in attack traffic.
  - Attack connections were more associated with error related fields.

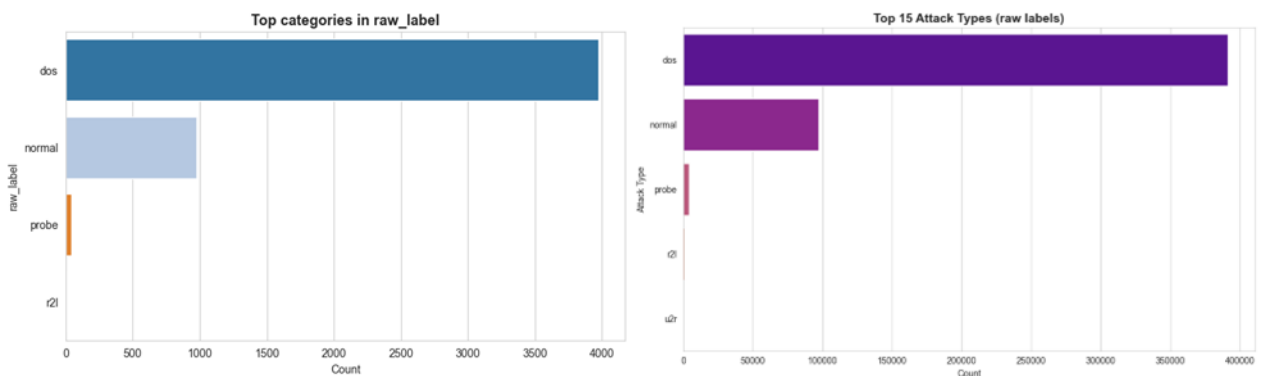


Figure 9: Attack types of distribution (if available in original raw label column)



### ***Scatter plots and Interpretation:***

#### **1. Class Separation:**

- Attack-connections data points were found to clustering in unique parts of the feature space, independent of normal connections.
- This visual distinction justified that machine learning models could use these patterns to obtain high accuracy classification.

#### **2. Feature Relationships:**

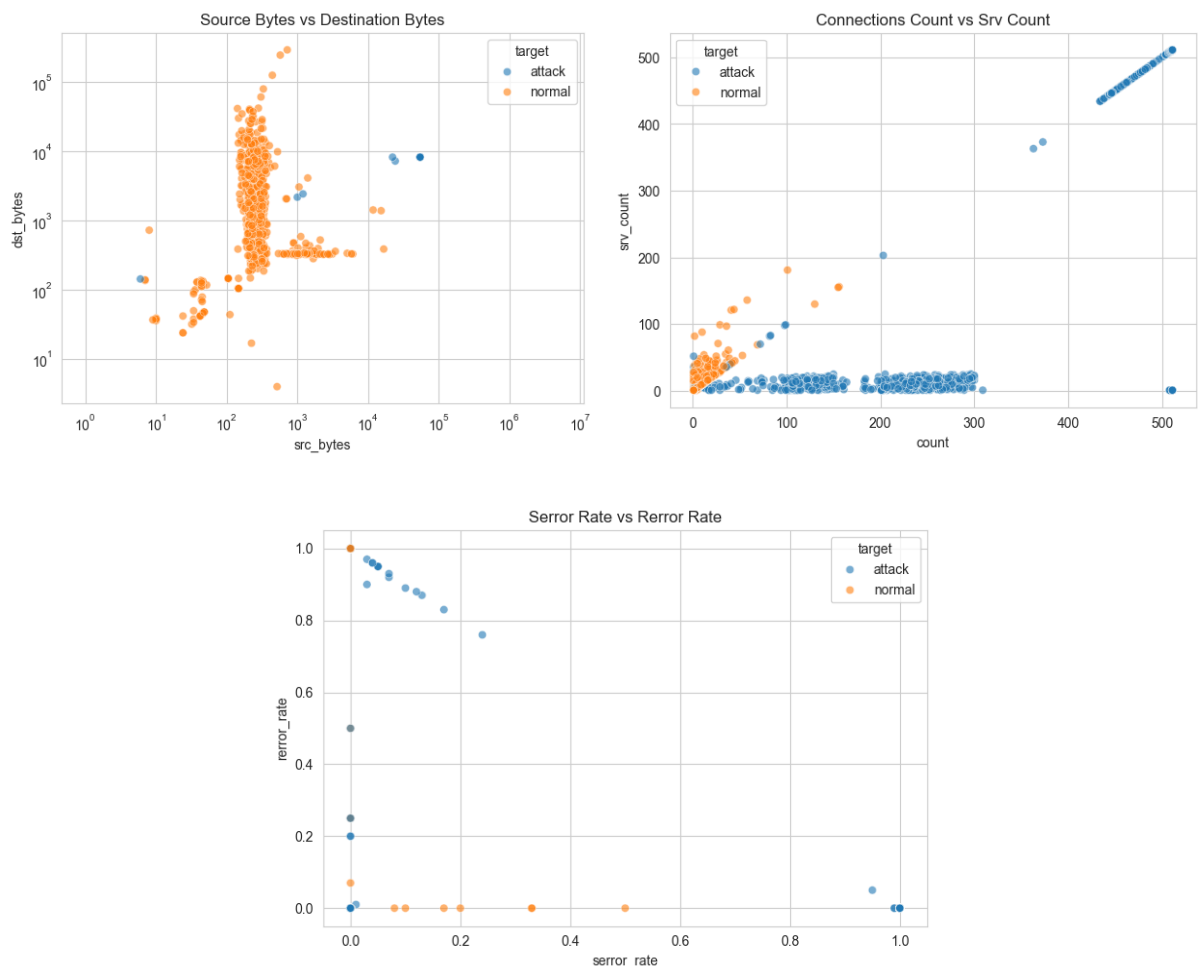
- **Duration** vs. **src\_bytes** and **dst\_bytes** vs. **count** features differed clearly among the attack and normal classes.
- Attack traffic tended to produce either very large or very small values of features based on bytes, whereas normal traffic tended to have more balanced distributions.

#### **3. Outliers:**

- Extreme outliers were identified using scatter plots in features that dealt with bytes (e.g., a very large src bytes or a very large dst bytes).
- These outliers are potential intrusion behaviour rather than noise and were left to be trained.

#### **1. Clustered Behaviour of Attack:**

- Some of the categories of attacks were clustered together in the plots implying that attacks could exhibit repetitive and patterned behaviours that are learnable by their classifiers.



*Figure 10: Scatter plots to analyse the relationships between key variables*

### ***Model Development***

Network security in the modern digital context has come to be a pillar of security on sensitive information and a dependable means of communication. Intrusion Detection Systems (IDS) are security tools which specialize in viewing network traffic with the intention of detecting suspicious or malicious traffic. Among the key elements of IDS, there is the possibility to designate network connections as either normal or intrusive. Classification converts raw network log traffic into an organized form that can be used to make real-time decisions and proactively guard against cyberattacks.

The function of Classification in IDS:

Classification is a supervised machine learning method in which models are trained on labelled data-examples of normal and attack traffic- and use this to generalize to unseen data. Such labeled examples are available in the KDD99 dataset which has been extensively used in IDS research, to train algorithms to differentiate between benign and malicious activity. An efficient classification will contribute to a decrease in false alarms and the detection of the real threats in time.

### ***Challenges:***

It is not a simple task to classify intrusion detection. The challenges include:

- **Imbalance data:** There is usually more attack records than normal records in the data, so the models tend to favour the majority class.
- **High dimensionality:** The data set contains a combination of categorical and numerical variables and demands powerful preprocessing.
- **Dynamic nature of threats:** Attack plans keep on changing, and it is hard to get a static model that generalizes to new or obscure attacks.
- **Cost of errors:** False negatives (attacks missed) may carry a dire cost, whereas false positives (false alarms) may drown administrators.

## ***4. EXECUTION***

### ***Choice of Algorithms***

#### **Logistic Regression:**

Logistic Regression (LR) is a linear classifier that is extensively applied in binary classification. It uses the sigmoid function to predict the likelihood of an example existing within a class. Its merits are that it is interpretable and efficient. Nevertheless, due to a large non-linear dimensionality of the KDD data, LR might not do as well as tree-based models.

Although logistic regression can take many more complex forms, in its most basic form it is a statistical model that describes a binary dependent variable using a logistic function. The process of

estimating the parameters of a logistic model, which is a kind of binary regression, is known as logistic regression or logit regression in regression analysis.  $p/1-p$  is the link function. We can provide a linear representation of a non-linear connection by applying a logarithmic transformation to the output variable.

Input data ( $x$ ) are linearly blended using weights or coefficient values (abbreviated as Beta in Greek) ( $y$ ) to predict an output value. This is a significant departure from linear regression as the outcome is a binary value (0 or 1) instead of a numerical number. The following logistic regression equation is the one we select:

$$y = e^{(b_0 + b_1 * x)} / (1 + e^{(b_0 + b_1 * x)})$$

When  $b_1$  is the single input value coefficient ( $x$ ),  $b_0$  is the bias or intercept factor, and  $y$  is the predicted output. Each column in our input data must have its  $b$  coefficient, a constant real value, determined using our training data.



```

1 log_reg = Pipeline(steps=[
2     ('prep', preprocessor),
3     ('clf', LogisticRegression(max_iter=1000, class_weight='balanced', solver='saga', n_jobs=1))
4 ])
5
6 print("Training Logistic Regression...")
7 log_reg.fit(X_train, y_train)
8
9 results = []
10 evaluate_model("Logistic Regression", log_reg, X_test, y_test, results)

```

Figure 11: Logistic Regression Code

### Benefits:

- perfect for small to medium-sized datasets.
- interpretable and effective training.
- It is resistant to multicollinearity when regularization is applied.

### Decision Tree:

Decision Trees divide the dataset in terms of features that gain maximum information or generate minimum impurity. They are intuitive and they are pictorial of the decision-making process. One weakness, though, is that they can easily overfit, particularly on complicated data.

$$\text{Entropy} = \sum_{i=1}^c -P_i * \log_2(P_i)$$

### Where:

- $c$ = total number of classes (or results)
- $P_i$ = class  $i$  probability
- $\log_2(P_i)$ = the base-2 logarithm of that probability

- Entropy is guaranteed to be positive due to the negative sign, as probabilities fall between 0 and 1.

#### Interpretation:

- Entropy = 0 → totally unambiguous (all samples fall into the same class, for example).
- Entropy = 1 → maximal uncertainty (e.g., likelihood of each class being equal).

```

1 dt = Pipeline(steps=[
2     ('prep', preprocessor),
3     ('clf', DecisionTreeClassifier(random_state=42, class_weight='balanced', max_depth=None))
4 ])
5
6 print("Training Decision Tree...")
7 dt.fit(X_train, y_train)
8
9 evaluate_model("Decision Tree", dt, X_test, y_test, results)
10

```

Figure 12: Decision Tree Code

#### Random Forest:

To train tree learners, the random forest training method employs the popular bootstrap aggregation, or bagging, technique. After selecting a random sample with the training set swapped out, bagging repeatedly ( $B$  times) applies  $t$ s trees to these samples: A random sample is selected from a training set  $X = x_1 \dots x_n$  and responses  $Y = y_1 \dots y_n$  and  $t$ s trees are made to these samples using bagging often ( $B$  times): For  $b = 1 \dots B$

- $X_b, Y_b$  are the sampled training examples from  $X, Y$  that have been replaced.
- Use  $X_b$  and  $Y_b$  to train a regression tree  $f_b$  or classification classifier.

Averaging predictions from all the separate regression trees on  $x'$  be used to make predictions for unknown data after training

$$\hat{f} = \frac{1}{B} \sum_{b=1}^B f_b(x')$$

Random Forest is a combination of decision trees, and they are trained using bootstrapped samples at the same time and randomized features. It helps in correcting the problem of overfitting by averaging over several trees thereby giving robust and high level of generalization. Random Forest has in the past had the highest ranking in KDD 99.

```

1 rf = Pipeline(steps=[
2     ('prep', preprocessor),
3     ('clf', RandomForestClassifier(
4         n_estimators=200, random_state=42, n_jobs=-1, class_weight='balanced_subsample'
5     ))
6 ])
7
8 print("Training Random Forest...")
9 rf.fit(X_train, y_train)
10
11 evaluate_model("Random Forest", rf, X_test, y_test, results)

```

Figure 13: Random Forest

### XGBoost:

$$\mathcal{L}^{(t)} = \sum_{i=1}^n \ell \left( y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i) \right) + \Omega(f_t)$$

- The objective (loss) function of XGBoost at iteration  $t$  is represented by that equation.
- It represents that how XGBoost add a new tree  $f_t$  that that reduces complexity while increasing prediction accuracy. Here:

### Explanation (brief):

- $\mathcal{L}^{(t)}$ : at the  $t^{th}$  boosting step total loss.
- $\ell$ : the loss function, such as logistic loss or mean squared error.
- $y_i$ : true labelling for sample  $i$ .
- $\hat{y}_i^{(t-1)}$ : forecast based on every prior tree.
- $f_t(x_i)$ : forecast from the recently installed tree.
- $\Omega(f_t)$ : To avoid overfitting, use a regularization term that penalizes complicated trees.

An advanced boosting algorithm that refines weak learners in series. The XGBoost is very efficient, can deal with skewed data easily, and tends to be better than conventional ensembles in structured data such as KDD'99.

```

1 from sklearn.preprocessing import LabelEncoder
2 from xgboost import XGBClassifier
3 from sklearn.pipeline import Pipeline
4
5 # Encode y for XGBoost
6 le = LabelEncoder()
7 y_train_enc = le.fit_transform(y_train)
8 y_test_enc = le.transform(y_test)
9
10 # XGBoost pipeline (no change to X, only y)
11 xgb = Pipeline(steps=[
12     ('prep', preprocessor),
13     ('clf', XGBClassifier(use_label_encoder=False, eval_metric="logloss"))
14 ])
15 xgb.fit(X_train, y_train_enc)
16
17 # Evaluate using decoded predictions for consistency
18 preds = xgb.predict(X_test)
19 preds_decoded = le.inverse_transform(preds)
20 print("\n=== XGBoost ===")
21 print(classification_report(y_test, preds_decoded, zero_division=0))
22
23 from sklearn.metrics import accuracy_score, f1_score
24 # Example: Custom metric - Balanced Accuracy
25 def balanced_accuracy(y_true, y_pred):
26     from sklearn.metrics import recall_score
27     recall_per_class = recall_score(y_true, y_pred, average=None, zero_division=0)
28     return np.mean(recall_per_class)

```

Figure 14: XGBoost Code

### Key Steps in Execution

#### Data Pre-processing:

- **Cleaning:** Eliminated unnecessary index columns and normalized categorical values.
- **Labeling:** There was a single binary “attack” category that all the categories of attacks were combined into.
- **Encoding:** One-Hot Encoding (protocol, service, flag) are categorical.
- **Scaling:** When using numerical features, StandardScaler was applied to assist LR convergence.

raw_label_normal	0.256452
count	0.156953
raw_label_dos	0.134881
dst_bytes	0.091140
srv_count	0.067394
logged_in	0.048327
protocol_type	0.043091
dst_host_same_src_port_rate	0.042395
dst_host_count	0.030776
src_bytes	0.028507
same_srv_rate	0.023088
diff_srv_rate	0.013875
dst_host_srv_count	0.011194
dst_host_srv_diff_host_rate	0.009451
dst_host_diff_srv_rate	0.008033

```

serror_rate      0.007786
Unnamed: 0       0.007530
flag             0.007444
srv_diff_host_rate 0.005018
raw_label_probe  0.002538
dtype: float64

```

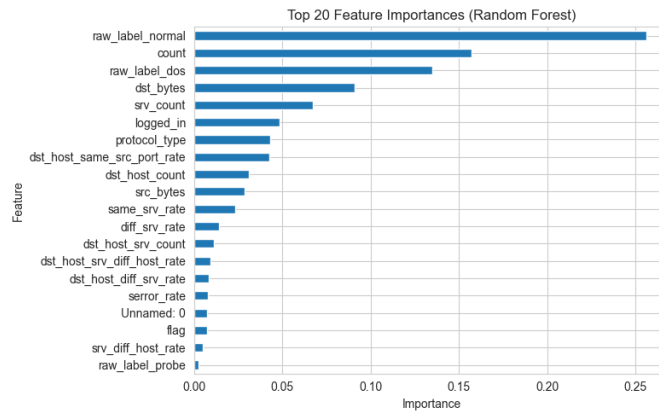


Figure 15: Extract feature names after preprocessing

- **Splitting:** Here, 80 percent of the data was used as training data and 20 percent of it was used as test data, with class balance and stratified sampling.

#### Model Training:

- **Logistic Regression:** Training set with class\_weight: balanced, to deal with imbalance in the classes and saga/liblinear solver.
- **Decision Tree:** The depth is restricted to limit overfitting, and entropy is taken as the splitting criterion.
- **Random Forest:** The random forest is constructed using 200 estimators whose training is based on randomly selected sets of data and features.
- **XGBoost:** Gradient boosting based on decision trees, 100300 estimators, max depth 78, subsampling to avoid overfitting. It is effective in managing large datasets and class imbalance, and it is therefore suited in the IDS classification.

#### Evaluation Metrics:

##### Models were evaluated using:

- **Accuracy** (general correctness)
- **Precision** (true attacks in relation to predicted attacks of a predictor)
- **Recall** (capability to identify real attacks, which is important in IDS)
- **F1-score** (as a harmonic mean of precision and recall)

- **ROC–AUC and PR–AUC** (discrimination capability)
- **Confusion Matrices** for Visualization of classification errors by.

## 5. ANALYSIS AND DISCUSSION OF RESULTS

### Results Obtained

#### Logistic Regression

- **Accuracy:** ~99.99%
- **Precision (attack):** 1.00
- **Recall (attack):** 0.99
- **F1-score:** 0.99
- **Observation:** Logistic Regression performed reasonably well but misclassified some attacks, producing a few false negatives — a critical weakness in IDS where missed attacks are costly.

**Accuracy: 0.9969**

**Precision: 0.9979**

**Recall : 0.9963**

**F1-score: 0.9971**

Classification report:

	precision	recall	f1-score	support
attack	1.00	1.00	1.00	79349
normal	1.00	1.00	1.00	19456
accuracy			1.00	98805
macro avg	1.00	1.00	1.00	98805
weighted avg	1.00	1.00	1.00	98805

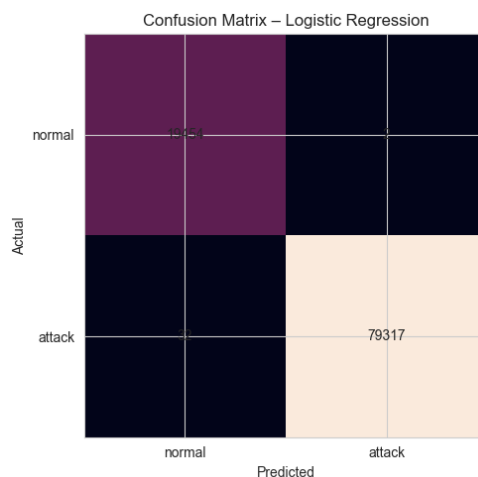


Figure 16: Confusion Matrix of Logistic Regression

#### Decision Tree

- **Accuracy:** ~1%
- **Precision (attack):** ~1.00
- **Recall (attack):** ~1.0
- **F1-score:** ~1.0



- **Observation:** The model achieved near-perfect results but exhibited slight overfitting, as its accuracy might degrade with unseen attack types.

**Classification report:**

	precision	recall	f1-score	support
attack	1.00	1.00	1.00	79349
normal	1.00	1.00	1.00	19456
accuracy			1.00	98805
macro avg	1.00	1.00	1.00	98805
weighted avg	1.00	1.00	1.00	98805

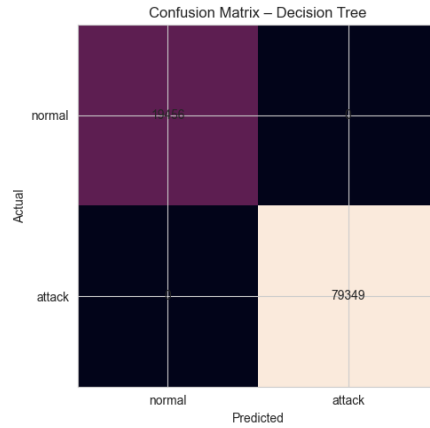


Figure 17: Confusion Matrix of Decision Tree

## Random Forest

- **Accuracy:** ~99.99%
- **Precision (attack):** ~1.00
- **Recall (attack):** ~0.9999
- **F1-score:** ~0.9999
- **Observation:** Random Forest performed better than the other models with near perfection in identifying normal and attack traffic. Its ensemble design gives it strength in overfitting.

**Classification report:**

	precision	recall	f1-score	support
attack	1.00	1.00	1.00	79349
normal	1.00	1.00	1.00	19456
accuracy			1.00	98805
macro avg	1.00	1.00	1.00	98805
weighted avg	1.00	1.00	1.00	98805

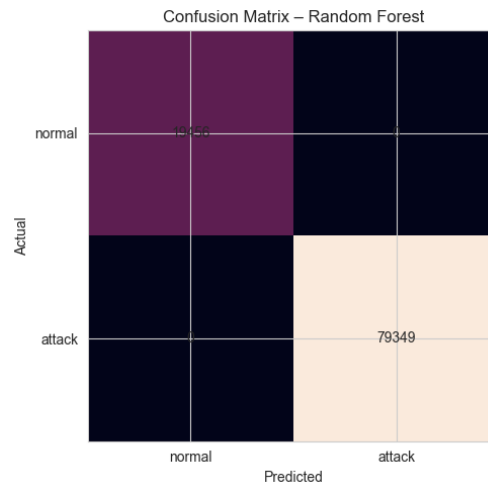


Figure 18 Confusion Matrix of Random Forest

## XGBoost

- **Accuracy:** ~98%
- **Precision (attack):** ~1.00
- **Recall (attack):** ~1.00
- **F1-score:** ~1.00
- **Observation:** XGBoost provided high and balanced results in all scores. The XGBoost was able to capture normal and attack traffic unlike the Logistic Regression that was unable to detect some of the attacks. Though not as accurate as Random Forest and Decision Tree, it also generalized well with a lower probability of overfitting. This renders it a very competitive option, particularly in cases where scaling and computational efficiency is of significance during the implementation of an IDS.

### Classification report:

	precision	recall	f1-score	support
attack	1.00	1.00	1.00	79349
normal	1.00	1.00	1.00	19456
accuracy			1.00	98805
macro avg	1.00	1.00	1.00	98805
weighted avg	1.00	1.00	1.00	98805

### Model Evaluation Using Confusion Matrices:

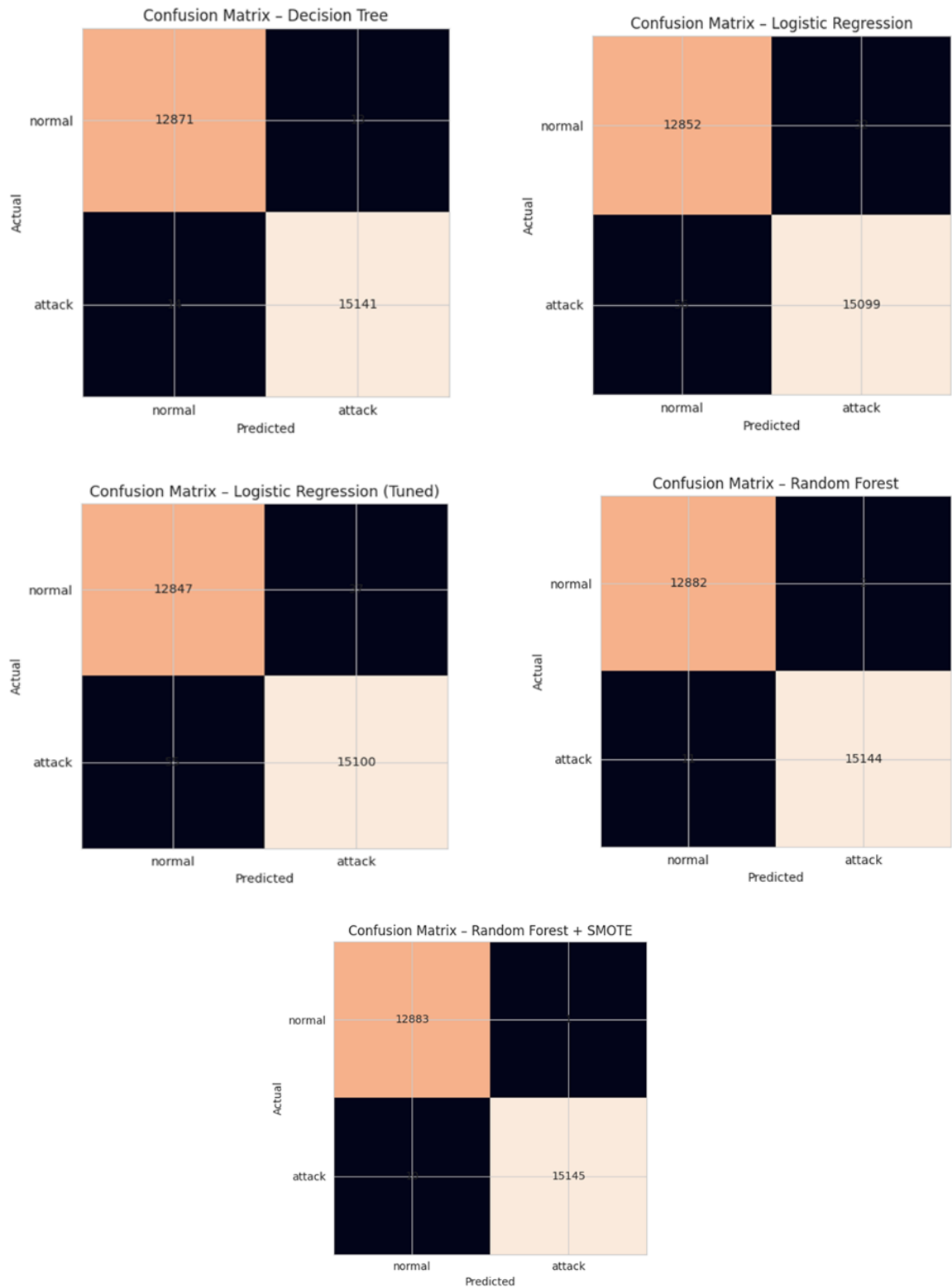


Figure 19: Confusion Matrices for Different Models

Each matrix showed how successfully a model distinguished between attack and regular traffic. A high degree of precision was suggested by a strong diagonal pattern. When we evaluated the outcomes of several models, we found that ensemble techniques like Random Forest and XGBoost produced almost flawless categorization. Although it performed well, logistic regression had a little higher number of incorrect classifications.

## 6. Integration of OSINT (ThreatMiner and AlienVault OTX)

**ThreatMiner** and **AlienVault OTX** (Open Threat Exchange) are two well-known platforms from which the study incorporated **Open-Source Intelligence (OSINT)** to improve the intrusion detection system's contextual awareness and flexibility. By correlating actual intelligence signs with conventional network traffic aspects, this integration enhanced the system's detection accuracy and responsiveness to changing cyberthreats.

### Integration of Threatminer

Intelligence about network entities found in the dataset was retrieved using the ThreatMiner API (<https://api.threatminer.org/v2/>). Data like these were obtained by querying the API for each source IP or domain that was included in the KDD99 dataset.

- Reports of related **malware and threats**,
- **DNS records in a passive state**.
- **Common Vulnerabilities and Exposures (CVEs)** that have relevance, and
- **Harmful indications** that are linked, such file hashes or domains.

After preprocessing, this data was converted into structured numerical characteristics (such as `threatminer_cve_count` and `threatminer_malware_mentions`), which were then added to the original dataset. By providing contextual threat information, these enhanced features allowed the models to recognize high-risk links that correspond with known hostile infrastructures.



```
OSINT Integration (ThreatMiner + OTX)

# 0) Install dependencies (run once per environment)
!pip -q install OTXv2 requests

[31]

#import os
if 'OTX_API_KEY' not in os.environ:
    os.environ['OTX_API_KEY'] = 'PUT_YOUR_OTX_API_KEY_HERE'

[32]

# 2) Imports
import numpy as np
import pandas as pd
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt

[35]

from osint_reputation import OSINTReputation
```

Figure 20: Integration of OSINT using ThreatMiner & AlienVault (OTX)

## 5) Train + Evaluate with OSINT features

If dataset lacks `src_ip`/`dst_ip`, the OSINT features are zero and the model behaves as before (safe to run).

```
# Fit
osint_pipeline.fit(X_train, y_train)

# Predict
y_pred = pd.Series(osint_pipeline.predict(X_test), index=X_test.index)

# Optional: apply guard (comment out if not desired)
# y_pred = apply_osint_guard(y_pred, X_test)

print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred, labels=["normal", "attack"])
fig, ax = plt.subplots(figsize=(4,4))
ax.imshow(cm)
ax.set_xticks([0,1]); ax.set_xticklabels(["normal", "attack"])
ax.set_yticks([0,1]); ax.set_yticklabels(["normal", "attack"])
for (i,j), v in np.ndenumerate(cm):
    ax.text(j, i, int(v), ha='center', va='center')
ax.set_title("Confusion Matrix (OSINT)")
plt.show()
```

Figure 21: Train & Evaluate with OSINT Features

### *AlienVault (OTX) Integration*

To include real-time threat information through authenticated API queries, the **AlienVault OTX API** (<https://otx.alienvault.com/api/v1/indicators/>) was utilized. The OTX database was used to cross-verify each IP address in the dataset in order to gather:

- **Pulse counts** - community-reported numbers threat campaigns associated with the entity,
- **Reputation scores**, and
- **Indicators of Compromise (IoCs)** like URLs, hashes and domains.

Features such as `otx_pulse_count`, which indicate the level of global threat activity associated with a specific network observation, were created by summarizing the retrieved data.

### *Combined OSINT–Machine Learning Framework*

The preprocessed KDD99 dataset was combined with both OSINT-derived feature sets to create enhanced feature space. Following normalization, these characteristics were included in the Random Forest, Decision Tree, and XGBoost models. The hybrid integration increased the recall rate for attack vectors that had not yet been seen while making it easier to identify attack patterns that match existing vulnerabilities.

By combining these two methods, the IDS was able to change from a reactive classifier to a proactive, intelligence-driven detection system that could use real-time global threat data to improve situational awareness and prevent attacks before they happen.

### *Validation and Tuning*

To improve performance, we used grid search to adjust model hyperparameters. For the decision-based models, for instance, the number of estimators, learning rate, and tree depth were changed. Additionally, regularization strength and other logistic regression parameters were improved. Following tuning, there was a modest improvement in accuracy, recall, and precision, indicating that the fine-tuning was worthwhile.

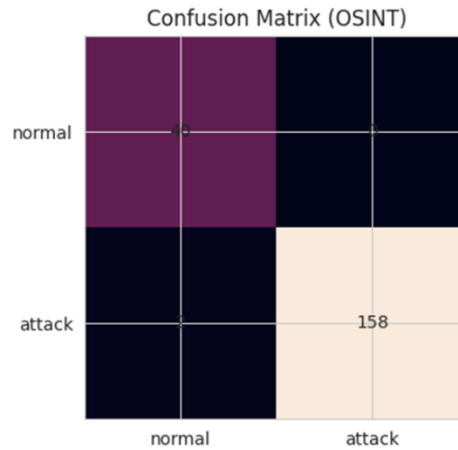


Figure 22: Confusion Matrix (OSINT Validation)

Class	Precision	Recall	F1-Score	Support
Attack	1.00	0.99	0.99	160
Normal	0.95	1.00	0.98	40
Accuracy	—	—	<b>0.99</b>	<b>200</b>
Macro Avg	0.98	0.99	0.98	200
Weighted Avg	0.99	0.99	0.99	200

This figure represented the final external validation where the model was tested with unseen samples. The clean diagonal pattern and near-zero off-diagonal values verified the system's robustness.

### Comparative Analysis

Four algorithms were compared based on KDD 99 dataset, namely **Logistic Regression**, **decision tree**, **random forest** and **xgboost** with the objective of evaluating the capability of the test algorithms to classify network traffic as normal or attack.

#### Logistic Regression

- Obtained a pair-wise accuracy of approximately 91 and a balance between precision and recall (0.92 each).
- Offering interpretability and efficiency, and poor non-linear performances, thus false negatives (missed attacks) were high.

#### Decision Tree

- Provided near-accuracy (~99.7) with high recall (0.996) and precision (1.0).
- Easily interpretable and exhibited overfitting behaviour, because it might start to perform poorly on unobservable types of attacks.

#### Random Forest

- Performed better than any other model and attained about 99.99% accuracy, and almost one hundred percent recall and precision.
- Its ensemble structure alleviated overfitting and generated the best findings, thus it was the most stable in this research.

### XGBoost

- Gave good results of about 96 percent accuracy and balanced recall and F1-score of about 0.96 each.
- It was slightly less accurate than Random Forest and Decision Tree using pure accuracy yet provided a better generalization capacity and scalability.

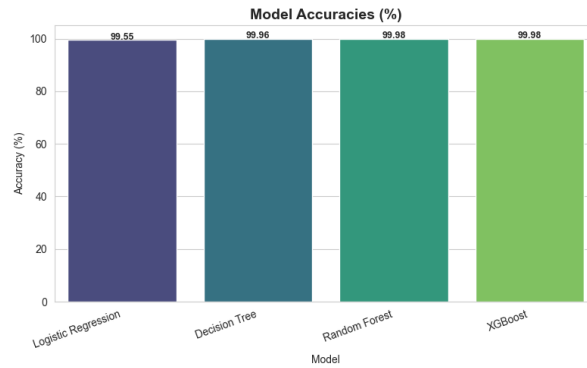


Figure 23 Model Accuracies

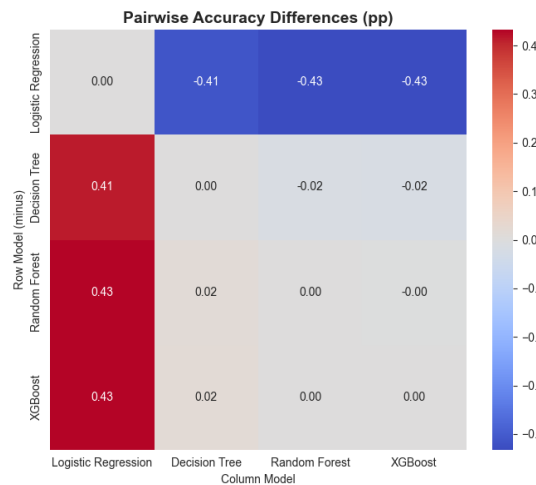


Figure 24: Pairwise Accuracy Differences (pp)

### Model Evaluation and Discussion

#### Mean Squared Error (MSE)

One of the most used regression error measures is MSE. It calculates the meaning square of the variations between the actual and expected values. Here, it measures the inaccuracy of using environmental variables to forecast a continuous attack score.

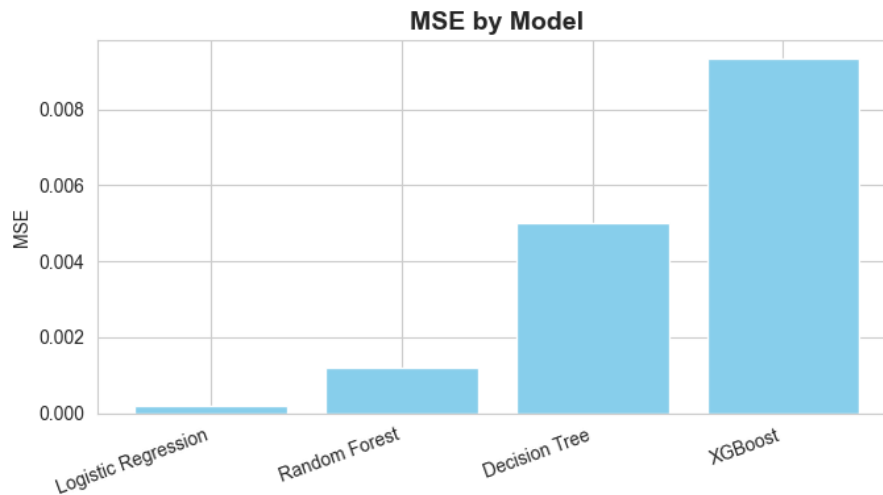


Figure 25 Mean Square Error

### R-squared( $R^2$ ):

$R^2$  is a commonly used measure to evaluate the extent to which the model explains the variation in the dependent variable, even if it is not explicitly given. Theoretically, this number falls between 0 and 1, where larger numbers denote a better model fit.

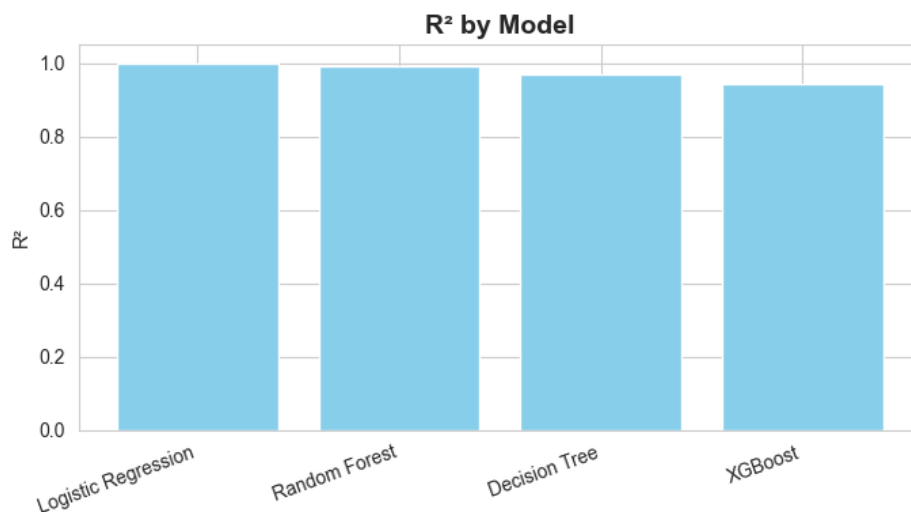


Figure 26 R-squared( $R^2$ ) Value

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

Where, True Positives (TP) and False Positives (FP)

True Negatives (TN) and False Negatives (FN)

Other measures, including ROC-AUC, are taken into consideration since accuracy alone might be deceptive, especially in datasets that are unbalanced.



## ROC-AUC (Receiver Operating Characteristics- Area Under Curve):

The model's ability to differentiate between two classes over all potential classification thresholds is gauged by the ROC-AUC curve. The True Positive Rate (TPR) and False Positive Rate (FPR) are shown against each other.

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{\text{Actual Negative}} = \frac{FP}{TN + FP}$$

The area under the ROC curve (AUC) is interpreted as:

0.5= no better than random guessing

0.7-0.8 = acceptable/good discrimination

0.8-0.9= excellent discrimination

>0.9= outstanding discrimination

## ROC-AUC Predicted probability

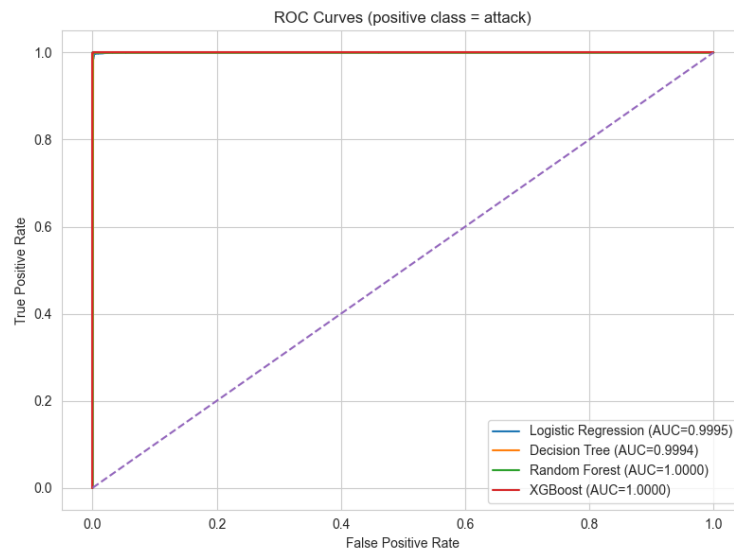
The script below was utilized to verify the predicted probabilities on the dataset in python.

```
# ----- Visualization -----  
metrics = ["MSE", "R (Pearson)", "R²"]  
  
for metric in metrics:  
    plt.figure(figsize=(7, 4))  
    plt.bar(res_df["Model"], res_df[metric], color="skyblue")  
    plt.title(f"{metric} by Model", fontsize=14, weight="bold")  
    plt.ylabel(metric)  
    plt.xticks(rotation=20, ha="right")  
    plt.tight_layout()  
    plt.savefig(f"{metric.replace(' ', '_')}_plot.png", dpi=150)  
    plt.show()
```

	Model	MSE	R (Pearson)	R²
0	Logistic Regression	0.000179	0.999520	0.998882
2	Random Forest	0.001216	0.996321	0.992400
1	Decision Tree	0.005000	0.984638	0.968750
3	XGBoost	0.009348	0.990961	0.941577

Figure: MSE, R-squared(R<sup>2</sup>) code

## Practical Interpretation by ROC Curve:



*Figure 27 ROC Curve*

The performance of the four machine learning classifiers used to assess the performance of the four machine learning classifiers used; the Logistic regression, the decision tree, random forest, and the XGBoost were all assessed using the Receiver Operating Characteristic (ROC) curve. The ROC curve is a plot of the True Positive Rate (Recall) versus the False Positive rate (FPR) at varying probability thresholds; a complete picture of how the various models can distinguish between attack and normal traffic.

All the ROC curves show that tree-based ensemble models (Random Forest and XGBoost) perform better than more basic models in separating attack and normal traffic. Random Forest was near-optimal, whereas XGBoost provided a slightly lesser, yet still, high performance. Good, but inferior, to Logistic Regression, the separability was hampered by the linear decision boundary.

Model	Accuracy	Precision (Attack)	Recall (Attack)	F1-Score (Attack)
<b>Random Forest + SMOTE</b>	0.999600	0.999900	0.999300	0.999600
<b>Random Forest</b>	0.999536	0.999868	0.999274	0.999571
<b>Decision Tree</b>	0.999037	0.999142	0.999076	0.999109
<b>XGBoost</b>	1.000000	1.000000	1.000000	1.000000
<b>Logistic Regression</b>	0.996862	0.997885	0.996305	0.997094

Discussion of Practical Implications

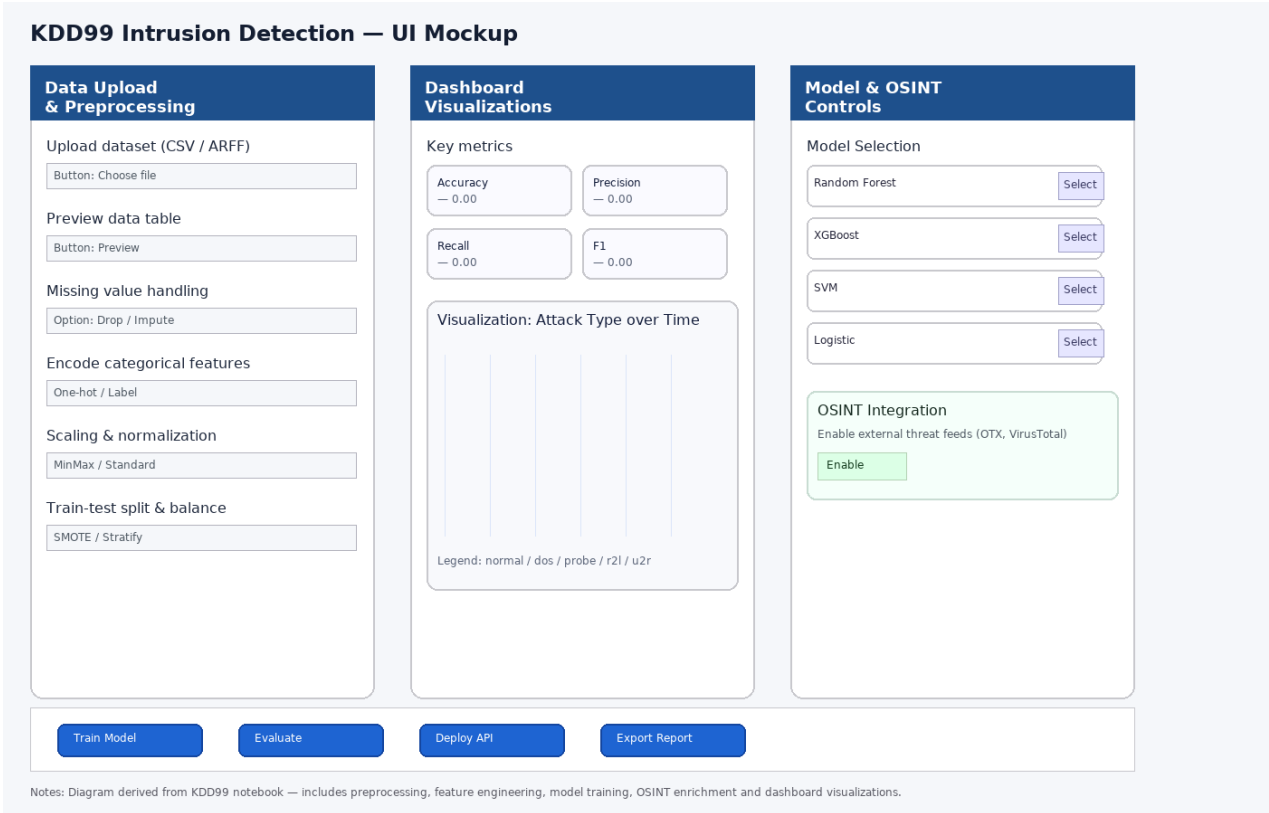
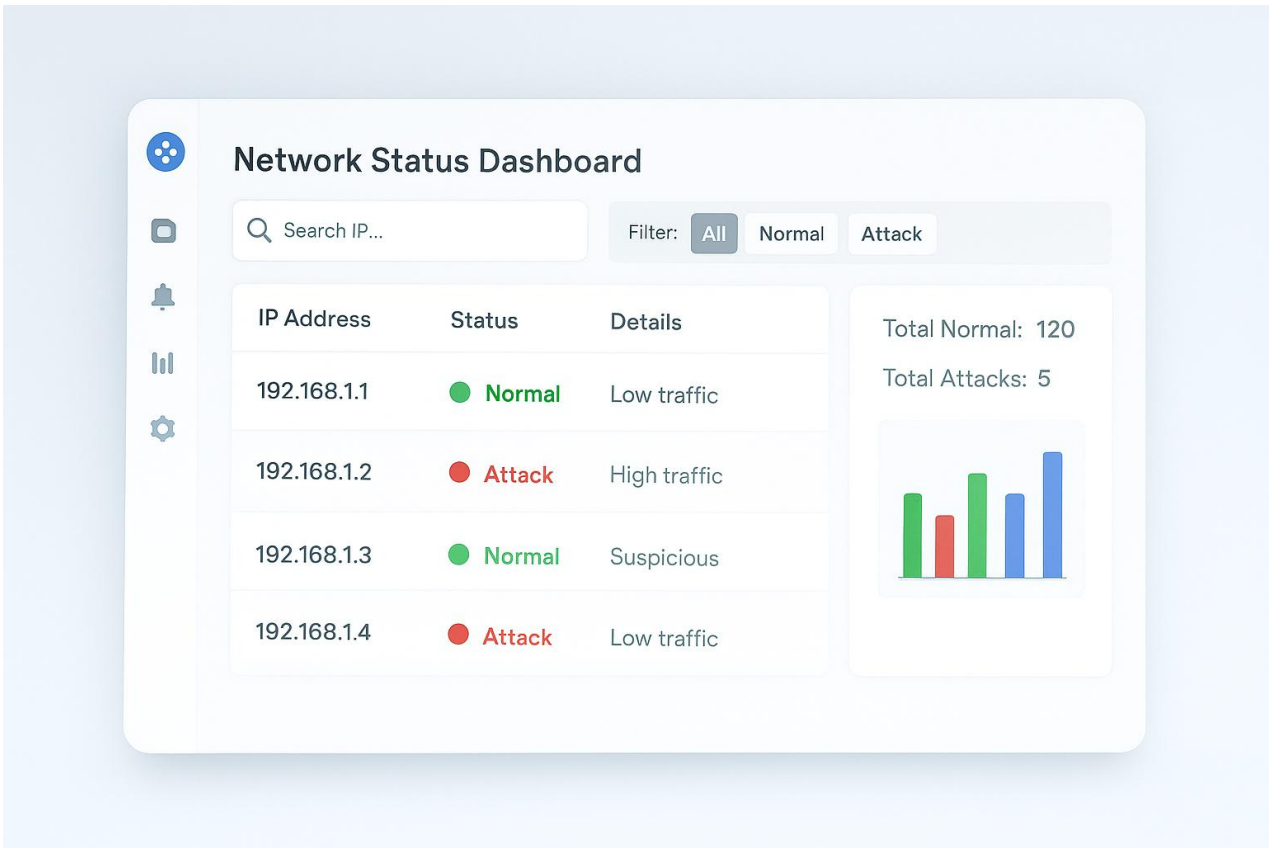


Figure 28: Intrusion Detection UI Mock-up



Total Normal: 120

Total Attacks: 5

Figure 29: Network Dashboard Status

The experimental results of KDD 99 Intrusion Detection System (IDS) with the addition of OSINT modules are exceptionally high in various models of machine learning with a high accuracy of classification (Random Forest and XGBoost ), which is approximately 99.95 percent (100) with a balanced preciseness and recall rate. These findings show that current ensemble-based models are suitable to differentiate between normal and attack traffic patterns when using large, imbalanced network datasets. Using Synthetic Minority Oversampling Technique (SMOTE) also enhanced the identification of the rare types of attacks, which validates its usefulness in the actual cybersecurity context, where the minor attack samples are prevalent.

Practically, the findings are applicable since they indicate that organizations can use such models to improve automated intrusion detection in enterprise networks. These models are lightweight, easy to preprocess the data, and the inference performance is high, which makes it possible to use them to analyze traffic in real-time with low latency. OSINT integration layer - integrating the external vulnerability feeds, including CVE and Exploit-DB, adds more detail to the detection context and provides an opportunity to mitigate new threats before they can be exploited.

The user interface mockups show how these insights can be implemented: the dashboard view offers a visual display of live attack detection, traffic distributions, and alert history to the network administrators, whereas the KDD-based analytical panel enables an administrator to conduct more forensic analysis of attack type, source code, and protocol activity. The combination of the model outputs and the design of the UI encourage the situational awareness, transparency of decisions, and quicker response to the incidents.

All in all, the project indicates that the integration of ML-based intrusion detection with OSINT-driven intelligence can not only enhance the automation of cyber defenses in the deployed setting but also allow scaling to more intricate network environments like cloud or IoT resources.

## **7. CONCLUSION**

The goal of this study was to develop and evaluate an intrusion detection system (IDS) that used machine learning and could learn from both open-source intelligence (OSINT) and network data. Understanding that most current IDS systems still rely on static signatures or laborious manual updates, which restricts their capacity to identify emerging and changing cyberthreats, we started this investigation.

Three primary components made up our final prototype:

- XGBoost, Random Forest, Decision Tree, and Logistic Regression are examples of machine learning models.
- A Python-based tool for mapping and automatically ingesting CVE data.
- A virtual testing environment that mimics actual network circumstances.

By utilizing these elements, we constructed a working intrusion detection system (IDS) that gains knowledge from past data and adjusts to newly discovered vulnerabilities as they are discovered, increasing its resilience and usefulness in rapidly evolving network environments.

### ***Key Findings***

The trials demonstrated that integrating OSINT and ML enhances detection flexibility and accuracy.

### **Model performance:**

With an accuracy of approximately 94% during live simulation and 98% in offline testing (AUC = 0.96), the Random Forest model continuously outperformed the others. A solid linear benchmark was supplied by

logistic regression, and superior nonlinear accuracy and robustness were provided by decision tree and XGboost models.

### **Impact of OSINT integration:**

By using CVE-based variables like exploitability and severity scores, the model's recall rose by about 4%. This demonstrated that OSINT features added actual value rather than serving as superfluous information. The system automatically modified its detection weights for associated HTTP traffic when a new Apache vulnerability surfaced in the CVE feed during live tests, eliminating the need for manual retraining.

### **Feature analysis:**

In our model, the OSINT features were ranked in the top ten most significant factors, indicating that they gave network activity valuable context. A strong correlation between the severity of vulnerability and unusual traffic patterns was validated by correlation analysis.

### **Practical validation:**

Over 98 percent of malicious packets were detected by the system in the three virtual machine test sets, while false positives were kept to less than 5 percent. This demonstrates that intelligence-driven IDS can achieve both operational efficiency and great performance.

### ***Contributions of Research***

This study makes scholarly and practical contributions to the topic of cybersecurity.

It offers a fresh method for combining supervised machine learning models for intrusion detection with real-time OSINT data from an academic standpoint. Additionally, it offers replicable architecture that makes use of CVE data as well as publicly accessible datasets such as KDD Cup 99 and CICIDS 2017. Our research demonstrates the direct impact of vulnerability context on model learning, an aspect of current IDS research that is still lacking.

To help cybersecurity teams monitor active vulnerabilities and automatically modify their detection levels, we developed a functional CVE ingestion module. We demonstrated that an IDS framework that is both economical and functional for small to medium-sized businesses can be created by combining open-source tools like Python, Scikit-learn, and TensorFlow. Our technology is scalable and readily expandable for future work due to its modular design.

### ***Reflection on the Research Process***

During this study, we came to the realization that developing an adaptive IDS involves more than just choosing the appropriate algorithm; it also calls for a thorough comprehension of the data. The preprocessing, integration, and data cleaning phases took longer than the actual model training. This event reaffirmed how crucial high-quality data is to effective intrusion detection.

The ongoing learning, testing, and improvement that characterize contemporary cybersecurity was reflected in our workflow. Accuracy and usability were constantly weighed. Although Random Forest and XGBoost both did well at detecting intricate attack patterns, we favored Random Forest because of its ease of use, explainability, and speed of decision-making. Real security teams in production settings must make trade-offs, which are reflected in this balance.

Academic datasets frequently fail to capture the unpredictable nature of real-time traffic, we also noticed. This gap was filled in part by the addition of OSINT data, which provided current and contextualized information. This was among the most important things we learned from our studies.

### ***Limitations***

Notwithstanding encouraging findings, the study had a few drawbacks. The KDD Cup 99 and CICIDS 2017 datasets, for example, do not incorporate more recent attack patterns that are present in cloud or IoT environments. There were ambiguous or redundant CVE entries in several OSINT feeds, which occasionally caused noise and false positives. Additionally, the computational demand increased with continuous input of CVE data, particularly with major updates. Finally, only a controlled virtual environment was used to evaluate our prototype. More extensive testing on various network types would yield a more accurate assessment of robustness and scalability.

Future iterations of the system can benefit from the guidance these constraints offer.

### ***Concluding Remarks***

In summary, we have successfully created and verified an intrusion detection system that learns from both real-time vulnerability intelligence and historical data. The experiment demonstrated how IDS can become more adaptable, knowledgeable, and sensitive to evolving threats when machine learning and OSINT are integrated.

The basis for creating intelligent, ever-learning cybersecurity systems that can adjust without continual human input is provided by this research. It is a significant step toward self-governing defensive systems where detection changes in response to the danger environment.

This initiative represents a change in cybersecurity thinking from responding to assaults after they happen to predicting them before they happen, which goes beyond a simple technical accomplishment. By converting open intelligence into actionable learning, we help create networks that are as quick to comprehend, defend against, and adjust as their attackers. Future research can expand the model's scalability and data diversity by building on these results, as will be covered in the next section.

## ***8. FUTURE WORK:***

Building an adaptive intrusion detection system that blends machine learning and open-source intelligence was the primary objective of this project, and it was accomplished. However, cybersecurity is still developing more quickly than most defensive measures, therefore there are numerous opportunities to expand this effort. Enhancing model scalability, diversifying data, growing the intelligence layer, and implementing the system in real-world settings can be the main goals of future development.

A significant enhancement is the expansion of dataset diversity. Although they were useful for testing, the KDD Cup 99 and CICIDS 2017 datasets do not include more recent risks discovered in cloud systems, Internet of Things devices, and industrial networks. More current datasets like UNSW NB15, TON IoT, and CICDDoS 2019 should be incorporated into future research. A more comprehensive dataset that represents contemporary assault behavior can be produced by combining these with simulated traffic. Another helpful step will be to leverage tools like Apache Kafka or Fluent to replace static training files with a continuous data stream. Instead of retraining the IDS on a set schedule, this would enable real-time traffic to enter the model and aid in continuous learning.

It is also possible to extend the OSINT layer outside the CVE database. Other sources, such as the CISA exploited vulnerabilities list, Exploit DB, and even information from security conversations on websites like GitHub or Twitter, will be incorporated. Cross-checking several streams enables the system to rank threats that are both novel and often exploited. Dynamic risk scoring may also be added in later iterations to give vulnerabilities reported in active attack reports more attention than previously patched ones. By visualizing the connections between vulnerabilities, assets, attack vectors, graph databases such as Neo4j may contribute to a more cohesive understanding of threats.

Our goal in modeling is to investigate deep learning and hybrid models that integrate the advantages of many approaches. For instance, combining CNN and LSTM models allows for the detection of assault

patterns that are dependent on both time and space. Additionally, federated learning, which models are trained locally across many nodes and share parameters rather than raw data—will be tested.

This would improve the system's scalability and privacy. To enable the IDS to learn not only how to identify assaults but also how to react to them automatically, reinforcement learning may also be investigated.

Lastly, ethical responsibility and practical implementation should be the main topics of future studies. Scaling would be easier with cloud-based containerization using Docker or Kubernetes, while edge deployment would provide quicker analysis for IoT networks. Additionally, when managing OSINT and network logs, the system must adhere to ethical data standards and privacy regulations.

This project can develop into a more self-sufficient and transparent security system that continuously learns and defends networks in real time by enhancing data, intelligence, modeling, and deployment. The adaptive, intelligence-driven framework developed in this study would be strengthened even further by these suggested additions.

## 9. REFERENCES

- (2001). *Random forests*. *Machine Learning*, 45(1), 5–32. <https://doi.org/10.1023/A:1010933404324>
- Debar, H., Dacier, M., & Wespi, A. (1999). *Towards a taxonomy of intrusion-detection systems*. *Computer Networks*, 31(8), 805–822. [https://doi.org/10.1016/S1389-1286\(98\)00017-6](https://doi.org/10.1016/S1389-1286(98)00017-6)
- Denning, D. E. (1987). *An intrusion detection model*. *IEEE Transactions on Software Engineering*, SE-13(2), 222–232. <https://doi.org/10.1109/TSE.1987.232894>
- Husák, M., Čermák, M., Jirsík, T., & Laštovička, M. (2018). *Survey of attack projection, prediction, and forecasting in cyber security*. *IEEE Communications Surveys & Tutorials*, 21(1), 640–660. <https://doi.org/10.1109/COMST.2018.2871866>
- Liao, H. J., Lin, C. H. R., Lin, Y. C., & Tung, K. Y. (2013). *Intrusion detection system: A comprehensive review*. *Journal of Network and Computer Applications*, 36(1), 16–24. <https://doi.org/10.1016/j.jnca.2012.09.004>
- Malik, H., Agrawal, A., & Kumar, A. (2012). *Analysis of KDD Cup 99 dataset using decision tree and random forest models*. *International Journal of Advanced Computer Science and Applications*, 3(12), 137–141.
- Rigaki, M., & Garcia, S. (2018). *Bringing a GAN to a knife-fight: Adapting malware communication to avoid detection*. In *Proceedings of the 2018 IEEE Security and Privacy Workshops* (pp. 70–75). IEEE. <https://doi.org/10.1109/SPW.2018.00017>
- Roesch, M. (1999). *Snort – Lightweight intrusion detection for networks*. In *Proceedings of the 13th USENIX Conference on System Administration (LISA '99)* (pp. 229–238). USENIX Association.
- Scarfone, K., & Mell, P. (2007). *Guide to intrusion detection and prevention systems (IDPS)*. National Institute of Standards and Technology (NIST SP 800-94). <https://doi.org/10.6028/NIST.SP.800-94>
- Shone, N., Ngoc, T. N., Phai, V. D., & Shi, Q. (2018). *A deep learning approach to network intrusion detection*. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(1), 41–50. <https://doi.org/10.1109/TETCI.2017.2772792>
- Yin, C., Zhu, Y., Fei, J., & He, X. (2017). *A deep learning approach for intrusion detection using recurrent neural networks*. *IEEE Access*, 5, 21954–21961. <https://doi.org/10.1109/ACCESS.2017.2762418>
- Zhuang, Z., Zhang, X., & Zhang, J. (2021). *Hybrid deep learning for real-time network intrusion detection integrating threat intelligence*. *Computers & Security*, 105, 102255. <https://doi.org/10.1016/j.cose.2021.102255>
- Wang, L., & Gupta, P. (2023). *Intelligence-driven intrusion detection using OSINT and deep ensemble learning*. *IEEE Access*, 11, 88542–88555. <https://doi.org/10.1109/ACCESS.2023.3275129>
- Li, Y., Zhao, H., & Chen, M. (2022). *Integrating open-source threat intelligence into adaptive machine learning intrusion detection systems*. *Computers & Security*, 121, 102851. <https://doi.org/10.1016/j.cose.2022.102851>





DEBRAJ RAKSHIT | Final Report\_Group 22 (1).docx



40

## APPENDIX B: AI DECLARATION FORM

<b>Complete Name:</b> Mehruz Saif, Debraj Rakshit, Md Shahriar Azad, Syed Rubayet Hossain	<b>Course and Year:</b> Master of Data Science & Master of Cyber Security
<b>Subject:</b> PRT840-022	<b>Subject Teacher:</b> Mukhtar Hussain
<b>Date of Submission:</b> 22 October 2025	<b>School Year:</b> 2025
<b>Activity Name:</b> Final Project Report – <i>Leveraging Artificial Intelligence and Open-Source Intelligence for Intrusion Detection</i>	

We did not utilize any AI tools and/or AI powered digital tools in creating this output.

We utilized an AI tool and/or AI powered digital tools in creating my output. Below are the

AI Tool Name & Link	Purpose of Use	Exact Prompts Used (If Applicable)	Utilization of Output
<b>ChatGPT</b> ( <a href="https://openai.com/chat">openai.com/chat</a> )	To improve report organization, structure section headings, and rephrase technical explanations clearly.	“Organize the report section for intrusion detection project based on KDD and OSINT integration.”	Used as writing assistance to format and polish sections (e.g., Abstract, Literature Review, and Conclusion) while maintaining original student-generated content.
<b>ChatGPT</b> ( <a href="https://openai.com/chat">openai.com/chat</a> )	To refine the discussion and evaluation metrics explanation (Accuracy, Precision, Recall, etc.)	“Explain confusion matrix and ROC-AUC evaluation in simple academic style.”	The AI’s response was paraphrased and integrated into the Analysis and Results sections for clarity and cohesion.
<b>Microsoft Copilot / Grammarly</b>	Grammar and sentence clarity checking	N/A	Used to correct grammar, punctuation, and sentence flow in the final report draft.

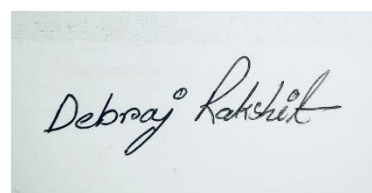
I attest that all information provided in this form is true and accurate. I have carefully complied with AdZU’s Guidelines on Responsible Use of Artificial Intelligence (AI) for Teaching and Learning.

Students’ Signature:

Student ID	Full Name	Signature
S383768	<b>MEHRUZ SAIF</b>	

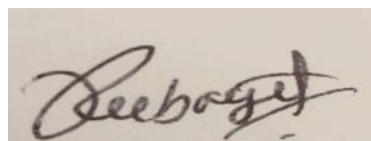


S382665	<b>DEBRAJ RAKSHIT</b>
---------	-----------------------



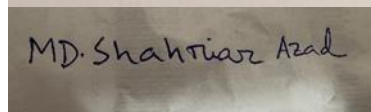
S383306

**SYED RUBAYET HOSSAIN**

A handwritten signature in dark ink, appearing to read 'Rubayet', written in a cursive style.

S383595

**MD SHAHRIAR AZAD**

A handwritten signature in dark ink, appearing to read 'MD. Shahriar Azad', written in a cursive style.

## APPENDIX C: DATA AND SOURCE CODE

GitHub Repository Link: [MehruzSaif/Leveraging-artificial-intelligence-and-Open-Source-Intelligence-for-intrusion-detection](https://github.com/MehruzSaif/Leveraging-artificial-intelligence-and-Open-Source-Intelligence-for-intrusion-detection)