Tunisian Republic
Ministry of Higher Education and Scientific Research
University of Sfax
National School of Engineers of Sfax

National Engineering Diploma in Computer Engineering

Report of AI and CyberSecurity Project

Defensive Application

Presented by

Gargouri Rahma

Kharrat Mariem

Aloulou Nihel

Charfi Safa

Academic Year: 2024-2025

Contents

List of Figures

List of Tables

Introduction

With the exponential growth of data exchanges over networks, cybersecurity has become a critical concern for organizations.

Cyberattacks are increasing in frequency and sophistication, and traditional intrusion detection systems struggle to keep pace with evolving threats. The need to develop more advanced solutions capable of efficiently detecting network anomalies and intrusions is more pressing than ever.

In this context, artificial intelligence, particularly machine learning (ML) and deep learning (DL), offers promising solutions. These technologies can be leveraged to detect unusual patterns in network traffic, which often signal potential attacks or intrusions. However, the large-scale collection of data introduces significant privacy concerns, which must be addressed within a secure framework.

This project seeks to explore and develop models for anomaly and intrusion detection, while ensuring data privacy through the use of federated learning (FL). Federated learning enables collaboration on shared models without the need to centralize sensitive data, thus preserving privacy while enhancing detection capabilities.

Chapter 1

Week 1: Business understanding

Plan

1.1 Problem Definition and Objectives

1.1.1 Introduction

In this section, we will explore the challenges associated with network anomaly and intrusion detection in the context of increasing network complexity and cyber threats.

We will define the core problems this project seeks to address, outline our objectives, and discuss the methodologies employed to develop robust detection models.

1.1.2 Problem Definition

The increasing complexity and volume of network traffic have made it essential to develop efficient systems for **network anomaly** and **intrusion detection**.

This project aims to address these challenges by building robust models that uses **machine learning (ML)**, **deep learning (DL)**, and **federated learning (FL)** techniques. Using benchmark datasets such as **UNSW-NB15** and **CICDDoS2019**, the objective is to detect abnormal behaviors and potential security threats in network traffic. The project employs the **CRISP-DM (Cross-Industry Standard Process for Data Mining)** framework to guide the development of these models, ensuring a systematic approach from understanding the problem to deploying the solution.

1.1.3 Objectives

The objective of this project is :

- **Comprehensive Problem Understanding**: Analyze the network traffic data to identify patterns of normal behavior and detect anomalies or intrusions. This involves a thorough exploration of the problem domain, relevant datasets, and methodologies.
- **Implementation of ML/DL Models**: Design, implement, and evaluate machine learning and deep learning models capable of effectively detecting anomalies and network intrusions.
- **Privacy-Preserving Detection with Federated Learning**: Investigate the role of federated learning in improving detection capabilities without compromising the privacy of the data involved.

1.1.4 Project Scope

The project adopts the **CRISP-DM methodology**, a widely used framework for data mining and machine learning projects. It includes the following stages:

- **Business Understanding**: Define the objectives, requirements, and success criteria for detecting anomalies and intrusions in network traffic.
- **Data Understanding**: Explore and analyze the UNSW-NB15 and CICDDoS2019 datasets to identify relevant features and patterns.
- **Data Preparation**: Clean, preprocess, and transform the data to ensure it is suitable for machine learning model development.
- **Modeling**: Implement machine learning and deep learning algorithms to build models capable of detecting network threats.
- **Evaluation**: Assess the performance of the models using metrics such as accuracy, precision, recall, and F1-score to ensure reliable detection.
- **Deployment**: Prepare the models for integration into real-world network systems to provide real-time anomaly detection and intrusion prevention.

1.1.5 Expected Outcomes and Success Criteria

1.1.5.1 Expected Outcomes

Our expected outcomes are:
Accurate and Robust Models : The project aims to develop machine learning and deep learning models that can accurately detect anomalies and security breaches in network traffic with high precision and minimal false alarms.
Federated Learning Integration: Explore the advantages of federated learning to build decentralized models that protect data privacy while maintaining strong detection capabilities.
Real-World Application: The models should be capable of deployment in practical network environments to assist in real-time monitoring and prevention of security threats.

1.1.5.2 Success Criteria

To say this project is a successful project, it need those criteria:
High Detection Performance: The models should achieve high accuracy in identifying anomalies, with low rates of false positives and false negatives.

Efficient Real-Time Deployment: The solution should be scalable and efficient enough to handle real-time network traffic in dynamic environments.
Data Privacy via Federated Learning: Successful integration of federated learning, demonstrating that it enhances privacy without sacrificing the effectiveness of the detection system.

1.2 Research and Literature Review

1.2.1 Introduction

This section will explore network anomalies and intrusions in depth, two types of events that can compromise network security. We will define these terms, compare them, and discuss detection techniques. Additionally, we will examine various network attacks and lay the groundwork for exploring machine learning applications in network security.

1.2.2 Network Anomaly and Network Intrusion

1.2.2.1 Anomaly Network

Definition

A network anomaly refers to an unusual event or behavior within a network that deviates from expected patterns of operation. These anomalies can be caused by benign factors, such as temporary traffic surges, or more concerning issues, like system errors or hardware failures.

Causes

- Hardware or software failures.
- Configuration issues or human errors.
- Sudden changes in usage patterns (a spike in traffic due to an unplanned event).
- Unusual behavior that may or may not be malicious.

Example A sudden increase in network traffic on a segment could be due to a configuration error, hardware failure, or even a massive software update.

1.2.2.2 Intrusion Network

Definition A network intrusion refers to a malicious and unauthorized attempt to access a network, its resources, or data, with the intent to steal, manipulate, corrupt, or disrupt services. An intrusion is always associated with a deliberate security breach.

Causes

- Cyberattacks by hackers to steal sensitive information.
- Malware propagation to disrupt or exploit a network.
- Exploitation of network vulnerabilities by malicious external or internal actors.

Example An attacker trying to illegally access a server to steal sensitive data or launch a denial-of-service (DoS) attack is considered a network intrusion.

1.2.2.3 Difference Between Network Anomaly and Network Intrusion

Table 1.1: Differences between Network Anomaly and Network Intrusion

Aspect	**Network Anomaly**	**Network Intrusion**
Origin	Can be benign or malicious	Always malicious or unauthorized
Cause	Failures, errors, usage variations, etc.	Deliberate attacks: hacking, malware, etc
Intent	Often unintentional or accidental	Always intentional with malicious goals
Example	Traffic surge due to hardware failure	Unauthorized access attempt on a server
Response	Analyze to diagnose and fix	Deploy security measures to stop the attack
Potential Impact	May be temporary and correctable	Can cause severe damage: data theft, corruption, downtime

1.2.3 Anomaly Detection And Intrusion Detection

1.2.3.1 Anomaly Detection

Definition Anomaly detection is the process of identifying unusual events, behaviors, or patterns in a system or network by comparing them to a normal operating model. It relies on data analysis techniques (statistics, machine learning, etc.) to monitor and detect these anomalies in real-time or retrospectively. Anomaly Detection and Identification of Deviations Anomaly detection involves continuously monitoring traffic and events on a network to identify deviations from the normal behavior model. Here's how it works and how it can reveal security or performance issues:

1. **Establishing the Normal Behavior Model** : The network is analyzed over a period of time to determine what constitutes normal operation. This includes:

- Typical traffic volumes (upload/download).
- Frequently contacted IP addresses.
- The protocols used and the frequency of access to resources.

2. **Real-Time Monitoring** : Once the model is established, detection systems continuously monitor the network. They compare new actions, requests, or behaviors to this normal model.

3. **Identifying Deviations** : As soon as deviant behavior is detected (such as an unusual traffic spike, a connection to a suspicious IP, or an attempt to access sensitive data), it is flagged as an anomaly. These deviations can take various forms:

- **Point anomalies** : A single isolated event that differs from the norm.
- **Contextual anomalies** : Abnormal behaviors in a specific context (e.g., abnormal access to a critical system at night).
- **Collective anomalies** : A set of actions that, when combined, reveal abnormal behavior.

Indicators of Security Breaches, Malicious Activities, or System Failures

1. **Security Breaches**: Detected anomalies can often indicate unauthorized intrusions into the network. For example, a sudden spike in connections from an unknown IP address might signal an attempt at a brute force attack. Similarly, unusual data transfers to external IP addresses may indicate data exfiltration.

2. **Malicious Activities** : Anomalies can also signal malicious behaviors, such as:

- **Denial of Service (DoS) attacks** : Abnormally high traffic volume attempting to overload the network.
- **Malware spread** : Unusual behavior by certain systems or users may signal infection by malware or ransomware.
- **Suspicious internal behavior** : Employees accessing resources they typically don't have access to.

3. **System Failures**: Anomalies may also reveal technical problems like hardware or software failures. For example:

- Bandwidth saturation or an increase in latency might signal system overload
- Repeated error logs could indicate imminent hardware failures (such as failing hard drives or overloaded servers).
- **Collective anomalies** : A set of actions that, when combined, reveal abnormal behavior.

1.2.3.2 Intrusion Detection Systems (IDS)

Definition Intrusion Detection Systems (IDS) are security systems designed to monitor network traffic and detect suspicious activities or signs of potential intrusion. Their role is to identify unusual behaviors or known threat patterns by analyzing the data flows traversing a network or between different systems. When suspicious activity is identified, an IDS generates alerts so that network administrators or security systems can investigate and take appropriate action.

Operation 1. **Continuous Monitoring of Network Traffic** :
IDS are configured to analyze all data packets traversing the network in real-time. They inspect not only the packet headers but also the content of the transported data to detect anomalies or malicious patterns.

2. **Detection of Suspicious Activities** : IDS can identify suspicious activities using two main approaches:

- **Signature-Based Detection** : This method compares network traffic with a database of known attack signatures (such as virus, worm, or DoS attack signatures). If a match is found, an alert is triggered.
- **Anomaly-Based Detection** : Here, the IDS creates a model of normal network behavior and triggers an alert when activity significantly deviates from this model. This allows for the detection of unknown threats or new forms of attacks.

3. **Alert Generation** :
When potentially malicious activity is identified, the IDS generates an alert. Alerts generally contain information such as the source and destination IP addresses, the type of threat detected, the time of the incident, and sometimes suggestions for action.

Types

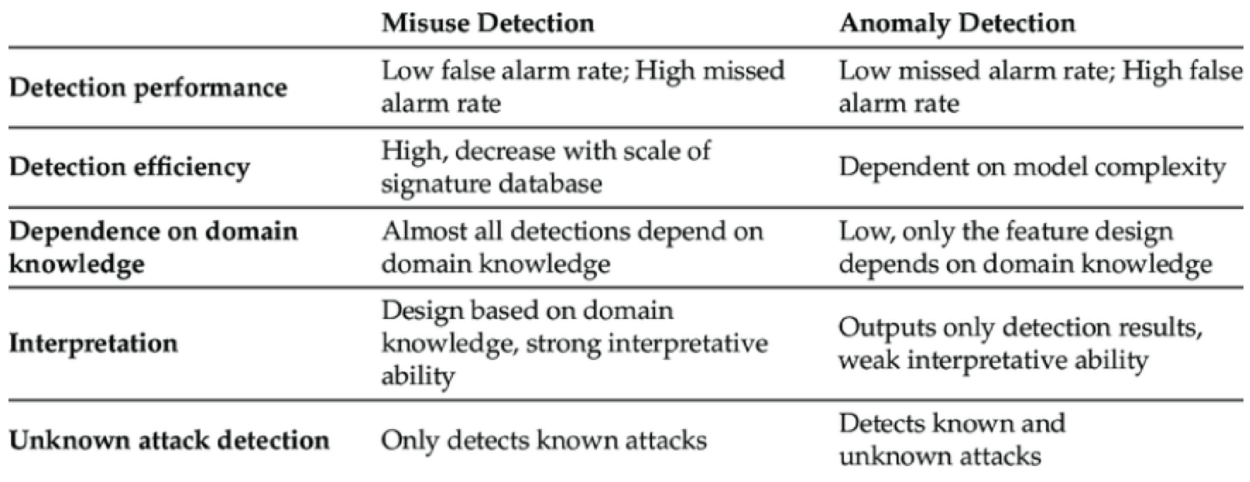
- **NIDS (Network Intrusion Detection System)** : The NIDS monitors overall network traffic. It is often placed at strategic points in the network (such as at the gateway or a central switch) and analyzes all traffic passing through these points to detect threats.

- **HIDS (Host-based Intrusion Detection System)** : The HIDS is installed directly on a host or individual machine (such as a server or computer) and monitors local activities, such as log files, modifications to critical files, and running processes.

Objectives

- **Attack Prevention** : Although an IDS does not directly prevent intrusions, it enables quick alerts to administrators so they can take measures to block or contain the attack.
- **Risk Reduction** : By detecting threats before they cause real damage, IDS help reduce the impact of cyberattacks on systems and data.
- **Security Enhancement** : Through the alerts generated, security teams can identify and correct vulnerabilities present in the network, thereby improving the overall security of the system.

Differences



	Misuse Detection	**Anomaly Detection**
Detection performance	Low false alarm rate; High missed alarm rate	Low missed alarm rate; High false alarm rate
Detection efficiency	High, decrease with scale of signature database	Dependent on model complexity
Dependence on domain knowledge	Almost all detections depend on domain knowledge	Low, only the feature design depends on domain knowledge
Interpretation	Design based on domain knowledge, strong interpretative ability	Outputs only detection results, weak interpretative ability
Unknown attack detection	Only detects known attacks	Detects known and unknown attacks

Figure 1.1: Differences between Intrusion and anomaly detection

1.2.4 Different Type of Network Aattack

1.2.4.1 Denial of Service (DoS) Attack And Distributed Denial of Service (DDoS)

Dos Attack 1. **Definition**

Denial of service (DOS) is a network security attack, in which, the hacker makes the system or data unavailable to someone who needs it. Hacker tries to make a network, system, or machine unavailable by flooding it with fake requests or traffic. This prevents real users from accessing it, causing anything from slowdowns to complete shutdowns.

2. **Types of DoS Attacks**

- **Volume-Based Attacks** : Volume-based attacks flood a network with too much data, overpowering its bandwidth and making the network unusable. Examples include UDP floods and ICMP floods. In a UDP flood, attackers send many UDP packets to random ports on a server, making the server busy trying to handle all these requests, which slows down or stops legitimate traffic.
- **Protocol Attacks** : Protocol attacks exploit weaknesses in network protocols to use up server resources. Examples are SYN floods and the Ping of Death. In a SYN flood, attackers send many SYN requests to a server but don't complete the handshake, leaving the server stuck with half-open connections. The Ping of Death involves sending oversized packets to crash or disrupt the target server.
- **Application Layer Attacks** : Application layer attacks target specific applications or services, causing them to crash or become very slow. Examples include HTTP floods and Slowloris. In an HTTP flood, attackers send many HTTP requests to a web server, consuming its resources. Slowloris keeps many connections to the server open by sending incomplete HTTP requests, preventing the server from handling new, legitimate requests.

1. **DDos Attack Definition** :

A distributed denial-of-service (DDoS) attack is a malicious attempt to disrupt the normal traffic of a targeted server, service or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic.
DDoS attacks achieve effectiveness by utilizing multiple compromised computer systems as sources of attack traffic. Exploited machines can include computers and other networked resources such as IoT devices.

2. **The Mechanics of a DDoS Attack** :

DDoS attacks are carried out with networks of Internet-connected machines.
These networks consist of computers and other devices (such as IoT devices)which have been infected with malware, allowing them to be controlled remotely by an attacker. These individual devices are referred to as bots (or zombies), and a group of bots is called a botnet.

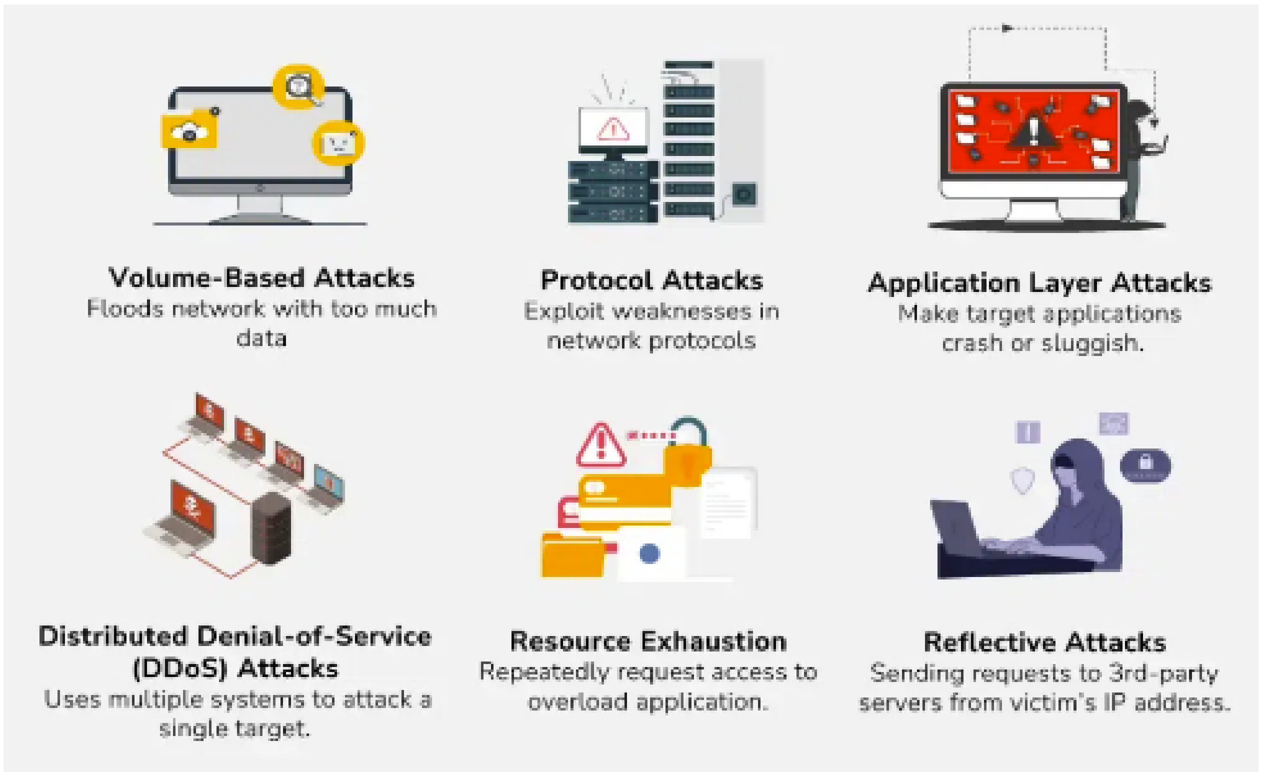

Volume-Based Attacks
Floods network with too much data
Protocol Attacks
Exploit weaknesses in network protocols
Application Layer Attacks
Make target applications crash or sluggish.
Distributed Denial-of-Service (DDoS) Attacks
Uses multiple systems to attack a single target.
Resource Exhaustion
Repeatedly request access to overload application.
Reflective Attacks
Sending requests to 3rd-party servers from victim's IP address.

Figure 1.2: Types of Dos Attacks

Once a botnet has been established, the attacker is able to direct an attack by sending remote instructions to each bot.
When a victim's server or network is targeted by the botnet, each bot sends requests to the target's IP address, potentially causing the server or network to become overwhelmed, resulting in a denial-of-service to normal traffic. Because each bot is a legitimate Internet device, separating the attack traffic from normal traffic can be difficult.

3. **Identifying a DDoS Attack** :

The most obvious symptom of a DDoS attack is a site or service suddenly becoming slow or unavailable. But since a number of causes — such a legitimate spike in traffic — can create similar performance issues, further investigation is usually required. Traffic analytics tools can help you spot some of these telltale signs of a DDoS attack:

- Suspicious amounts of traffic originating from a single IP address or IP range.
- A flood of traffic from users who share a single behavioral profile, such as device type, geolocation, or web browser version.
- An unexplained surge in requests to a single page or endpoint.
- Odd traffic patterns such as spikes at odd hours of the day or patterns that appear to

be unnatural

1.2.4.2 Probing/Scanning attacks

Site scanning and probing are methods used to gather information about the structure of a web application (e.g., pages, parameters, etc.) and the supporting infrastructure (e.g., operating system, databases, etc.). Targeted sites are scanned for known vulnerabilities in infrastructure software (such as IIS) as well as unknown vulnerabilities in the custom code developed for the specific target application. Probing is a more targeted approach, often used to test specific weaknesses or gather intelligence.

For bad actors, site scanning is conducted like military reconnaissance—a systematic examination of web applications to understand potential vulnerabilities and entry points. Alternatively, security teams may use scanners to identify potential issues, better protect data, and fortify their web applications. Basics Site scanning is an automated process that searches for security vulnerabilities in a web application. It can be a preemptive measure used by security professionals to detect issues.
Probing is manually conducted with the intent of digging deeper into specific areas of concern, providing a more granular view of potential security gaps.

The role of site scanning in cybersecurity is akin to a regular health screening. It serves as a first line of defense, identifying vulnerabilities before they become significant threats. The proactive measure can help safeguard information and ensure a seamless user experience for end-users.

Types of Site Scanning There are two common types of site scanning: vulnerability and port scanning. Vulnerability scanning is perhaps the most comprehensive form, designed to identify a wide range of threats and reporting back potential exposures.
Port scanning is like checking every door and window to ensure they're locked. It's a method that examines various network ports that might lack protection and be an entry point for an attacker. Network scanning expands on this approach by analyzing the broader network infrastructure, scrutinizing the the web presence for weaknesses.

1.2.4.3 Probing Techniques and Methodologies

Active and passive probing represent two sides of the same coin, each with distinct objectives and methods. Active probing is equivalent to knocking on doors to see who

answers. Passive probing is discreet, collecting data without direct interaction by using traffic analysis or monitoring system logs to infer the state of the system.
The use of automated probing tools has become more common in the security industry. These tools can swiftly scan through thousands of data points, identifying patterns and anomalies with machine-like precision.

1.2.4.4 Malware attacks

A malware attack is a common cyberattack where malware (normally malicious software) executes unauthorized actions on the victim's system. The malicious software (a.k.a. virus) encompasses many specific types of attacks such as ransomware, spyware, command and control, and more.
Criminal organizations, state actors, and even well-known businesses have been accused of (and, in some cases, caught) deploying malware. Like other types of cyber attacks, some malware attacks end up with mainstream news coverage due to their severe impact.
An example of a famous malware attack is the WannaCry ransomeware attack.

Types of Malware Attack Vectors There are three main types of malware attack vectors:

- **Trojan Horse** : This is a program which appears to be one thing (e.g. a game, a useful application, etc.) but is really a delivery mechanism for malware. A trojan horse relies on the user to download it (usually from the internet or via email attachment) and run it on the target.
- **Virus** : A virus is a type of self-propagating malware which infects other programs/files (or even parts of the operating system and/or hard drive) of a target via code injection. This behavior of malware propagation through injecting itself into existing software/data is a differentiator between a virus and a trojan horse (which has purposely built malware into one specific application and does not make attempts to infect others).
- **Worm** :Malware designed to propagate itself into other systems is a worm. While virus and trojan horse malware are localized to one infected target system, a worm actively works to infect other targets (sometimes without any interaction on the user's behalf).

Over the years, malware has been observed to use a variety of different delivery mechanisms, or attack vectors. While a few are admittedly academic, many attack vectors are effective at compromising their targets. These attack vectors generally occur over electronic communications such as email, text, vulnerable network service, or compromised website, malware delivery can also be achieved via physical media (e.g. USB thumb drive, CD/DVD, etc.).

Best Practices Against Malware Attacks The following best practices can help prevent a malware attack from succeeding and/or mitigate the damage done by a malware attack.

- **Continuous User Education** : Training users on best practices for avoiding malware (i.e. don't download and run unknown software, don't blindly insert "found media" into your computer), as well as how to identify potential malware (i.e. phishing emails, unexpected applications/processes running on a system) can go a long way in protecting an organization. Periodic, unannounced exercises, such as intentional phishing campaigns, can help keep users aware and observant. Learn more about security awareness training.
- **Use Reputable A/V Software** : When installed, a suitable A/V solution will detect (and remove) any existing malware on a system, as well as monitor for and mitigate potential malware installation or activity while the system is running. It'll be important to keep it up-to-date with the vendor's latest definitions/signatures.
- **Ensure Your Network is Secure** : Controlling access to systems on your organization's network is a great idea for many reasons. Use of proven technology and methodologies—such as using a firewall, IPS, IDS, and remote access only through VPN—will help minimize the attack "surface" your organization exposes. Physical system isolation is usually considered an extreme measure for most organizations, and is still vulnerable to some attack vectors.
- **Perform Regular Website Security Audits** : Scanning your organization's websites regularly for vulnerabilities (i.e. software with known bugs, server/service/application misconfiguration) and to detect if known malware has been installed can keep your organization secure, protect your users, and protect customers and visitors for public-facing sites.
- **Create Regular, Verified Backups** : Having a regular (i.e. current and automated) offline backup can be the difference between smoothly recovering from a destructive virus or ransomware attack and stressful, frantic scrambling with costly downtime/data-loss. The key here is to actually have regular backups that are verified to be happening on the expected regular basis and are usable for restore operations. Old, outdated backups are less valuable than recent ones, and backups that don't restore properly are of no value.

1.2.5 Reviewing some existing ML, DL, and FL in this domain

1.2.5.1 Machine Learning Approaches

1.2.5.2 Supervised Learning

Supervised learning algorithms have been widely applied in intrusion detection systems (IDS).

- **Support Vector Machines (SVM)**:
  - Highly effective for binary classification tasks.
  - Example: Mukkamala et al. (2002) used SVM for classifying normal vs. attack traffic.
  - Pros: Good generalization, effective in high-dimensional spaces.
  - Cons: Can be computationally intensive for large datasets.
- **Random Forests**:
  - Ensemble learning method using multiple decision trees.
  - Example: Zhang et al. (2008) applied Random Forests for multi-class intrusion detection.
  - Pros: Handles high-dimensional data well, robust to overfitting.
  - Cons: Can be computationally expensive, less interpretable than single decision trees.

1.2.5.3 Unsupervised Learning

Unsupervised learning is particularly useful for detecting novel or zero-day attacks.

- **K-means Clustering**:
  - Groups similar data points, useful for anomaly detection.
  - Example: Münz et al. (2007) used K-means for network traffic clustering.
  - Pros: Simple, scalable to large datasets.
  - Cons: Sensitive to initial centroid selection, struggles with non-spherical clusters.
- **Autoencoders**:

  - Neural network-based approach for dimensionality reduction and anomaly detection.
  - Example: Sakurada and Yairi (2014) applied autoencoders for unsupervised anomaly detection.
  - Pros: Can capture complex non-linear relationships.
  - Cons: Requires careful tuning, can be computationally intensive.

1.2.6 Deep Learning Approaches

Deep Learning has shown promising results in capturing complex patterns in network traffic.

1.2.6.1 Convolutional Neural Networks (CNN)

- Effective for spatial feature extraction from raw traffic data.
- Example: Wang et al. (2017) used 1D-CNNs for malware traffic classification.
- Pros: Automatic feature extraction, good at capturing local patterns.
- Cons: Requires large amounts of data, computationally intensive.

1.2.6.2 Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM)

- Well-suited for sequential data analysis in network traffic.
- Example: Kim et al. (2016) applied LSTM for intrusion detection in in-vehicle networks.
- Pros: Captures temporal dependencies, effective for time-series data.
- Cons: Can be slow to train, prone to vanishing/exploding gradients (mitigated in LSTM).

1.2.6.3 Deep Belief Networks (DBN)

- Composed of multiple layers of restricted Boltzmann machines.
- Example: Alom et al. (2015) used DBNs for intrusion detection in Internet of Things (IoT) networks.
- Pros: Can handle unlabeled data, good at feature extraction.
- Cons: Complex training process, less interpretable.

1.2.7 Federated Learning Approaches

Federated Learning is an emerging approach that addresses privacy concerns in collaborative learning.

1.2.7.1 Horizontal Federated Learning

- Aggregates model updates from multiple parties with the same feature space.
- Example: Preuveneers et al. (2018) applied horizontal FL for collaborative intrusion detection.
- Pros: Preserves data privacy, allows collaboration without data sharing.
- Cons: Communication overhead, potential for model poisoning attacks.

1.2.7.2 Vertical Federated Learning

- Combines data from parties with different feature spaces but overlapping samples.
- Example: Yang et al. (2019) proposed a vertical FL framework for cross-silo cybersecurity.
- Pros: Enables collaboration between organizations with complementary data.
- Cons: Complex coordination required, potential for information leakage.

1.2.8 Challenges and Future Directions

1.2.8.1 Challenges

- Adversarial attacks on ML models.
- Class imbalance in network security datasets.
- Real-time performance requirements.
- Interpretability of complex models.

1.2.8.2 Future Directions

- Explainable AI for intrusion detection.
- Integration of domain knowledge with ML/DL models.
- Adaptive learning systems for evolving threats.
- Secure and efficient federated learning protocols.

1.2.9 Conclusion

This section has provided an overview of the threats to network security. By differentiating between network anomalies and intrusions and presenting various attack types, we have highlighted the complexity of cybersecurity challenges. Intrusion detection systems are crucial for preventing and detecting attacks, but their effectiveness can be improved through machine learning techniques. Subsequent sections will delve deeper into machine learning, deep learning, and federated learning approaches to enhance network security.

1.3 Dataset Familiarization

1.3.1 Introduction

In cybersecurity, datasets like UNSW-NB15 and CICDDoS2019 are essential for developing and evaluating intrusion detection systems (IDS). They simulate real-world network traffic and various attack types, offering valuable resources for building machine learning models. This chapter provides an overview of these datasets, covering their structure, features, and applications, and highlights their importance in improving cybersecurity defense mechanisms.

1.3.2 UNSW-NB15 Dataset

1.3.2.1 Introduction

The UNSW-NB15 dataset was created to address the challenges of existing network intrusion detection datasets, which lacked diversity in modern attack types. It was developed using the**IXIA PerfectStorm** tool to simulate real modern-day network traffic, including both normal and malicious activities, across various types of attacks. The dataset comprises **100 GB** of raw traffic captured using the tcpdump tool, resulting in Pcap files that encompass a diverse range of network interactions. It features nine types of attacks. To generate a comprehensive feature set, the dataset utilizes tools such as **Argus** and **Bro-IDS**, producing a total of 49 features, which are documented in the **UNSW-NB15-features** file.

This comprehensive dataset aims to address the limitations of existing network intrusion detection datasets, providing a valuable resource for evaluating the performance of network intrusion detection systems (NIDS).

1.3.3 Dataset Features

The UNSW-NB15 dataset contains a total of 49 features divided into several categories:

1.3.3.1 Basic Features

These are attributes extracted from raw packets at the network level, including:

- **Source IP**: The IP address from which a packet is sent.
- **Destination IP**: The IP address to which a packet is directed.
- **Source Port**: The port number on the source system.
- **Destination Port**: The port number on the destination system.

1.3.3.2 Flow Features

Flow-based attributes represent network sessions, including:

- **Flow Duration**: The total duration of a flow in microseconds.
- **Total Fwd Packets**: The total number of packets sent from the source to the destination.
- **Total Bwd Packets**: The total number of packets sent from the destination to the source.

1.3.3.3 Time-Based Features

These attributes capture timing information such as:

- **Flow IAT Mean**: The mean inter-arrival time between packets in a flow.
- **Fwd IAT Min**: The minimum inter-arrival time between packets from the forward direction.

1.3.3.4 Content Features

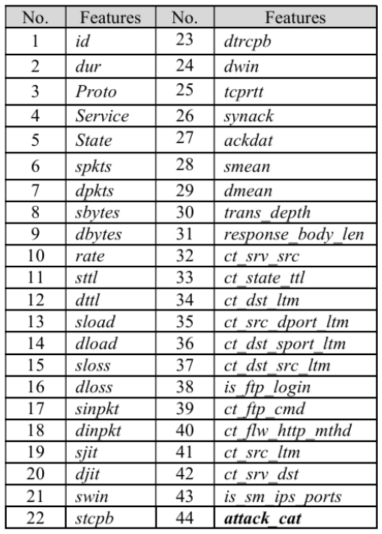
Content-based features relate to the payload and session behavior:

- **Fwd Header Length**: The total length of headers in the forward packets.
- **Bwd Header Length**: The total length of headers in the backward packets.

1.3.3.5 Additional Features

Other attributes include:

- **Protocol**: Identifies the transport protocol used (e.g., TCP, UDP).
- **Flag**: The status of the connection or error messages.



No.	Features	No.	Features
1	*id*	23	*dtrcpb*
2	*dur*	24	*dwin*
3	*Proto*	25	*tcprtt*
4	*Service*	26	*synack*
5	*State*	27	*ackdat*
6	*spkts*	28	*smean*
7	*dpkts*	29	*dmean*
8	*sbytes*	30	*trans_depth*
9	*dbytes*	31	*response_body_len*
10	*rate*	32	*ct_srv_src*
11	*sttl*	33	*ct_state_ttl*
12	*dttl*	34	*ct_dst_ltm*
13	*sload*	35	*ct_src_dport_ltm*
14	*dload*	36	*ct_dst_sport_ltm*
15	*sloss*	37	*ct_dst_src_ltm*
16	*dloss*	38	*is_ftp_login*
17	*sinpkt*	39	*ct_ftp_cmd*
18	*dinpkt*	40	*ct_flw_http_mthd*
19	*sjit*	41	*ct_src_ltm*
20	*djit*	42	*ct_srv_dst*
21	*swin*	43	*is_sm_ips_ports*
22	*stcpb*	44	***attack_cat***

Figure 1.3: Features of UNSW-NB15 Dataset

1.3.4 Attack Types

The dataset incorporates nine types of attacks, which are reflective of real-world cybersecurity threats:

1.3.4.1 Fuzzers

Attackers use fuzzers to discover vulnerabilities by sending random data to a target service or application.

1.3.4.2 Analysis

This includes various forms of reconnaissance, including port scanning and spam, to identify vulnerabilities.

1.3.4.3 DoS (Denial of Service)

DoS attacks aim to make a service or machine unavailable by overwhelming it with requests.

1.3.4.4 Exploits

Exploit attacks involve taking advantage of vulnerabilities in software, causing the system to malfunction or crash.

1.3.4.5 Generic

Generic attacks are aimed at cryptographic weaknesses, targeting all block ciphers.

1.3.4.6 Reconnaissance

This is the gathering of information about network services, ports, and vulnerabilities in preparation for an attack.

1.3.4.7 Shellcode

Shellcode refers to a small piece of code used as a payload in an exploit, often giving attackers control of the system.

1.3.4.8 Worms

Self-replicating malware that spreads across networks, typically without user intervention.

1.3.4.9 Backdoors

These attacks allow unauthorized access to systems, typically by installing hidden methods (backdoors) for attackers to bypass security and access the system later.

1.3.4.10 Dataset Composition

The UNSW-NB15 dataset is divided into training and testing sets. The training set contains **175,341 records**, while the testing set contains **82,332 records**. Each record in the dataset is labeled either as **normal** or **malicious** (attack).
Normal records: Represent benign network activities.
Attack records: Labeled with one of the nine attack types.

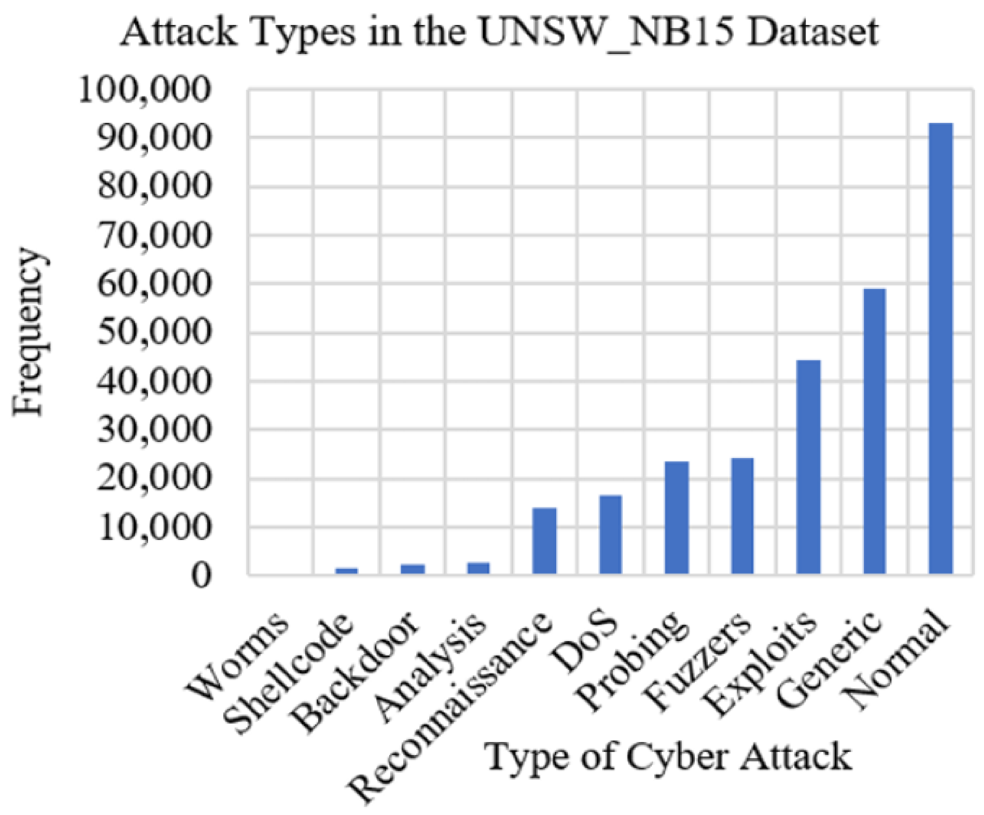


Figure 1.4: UNSW-NB15 frequency attack

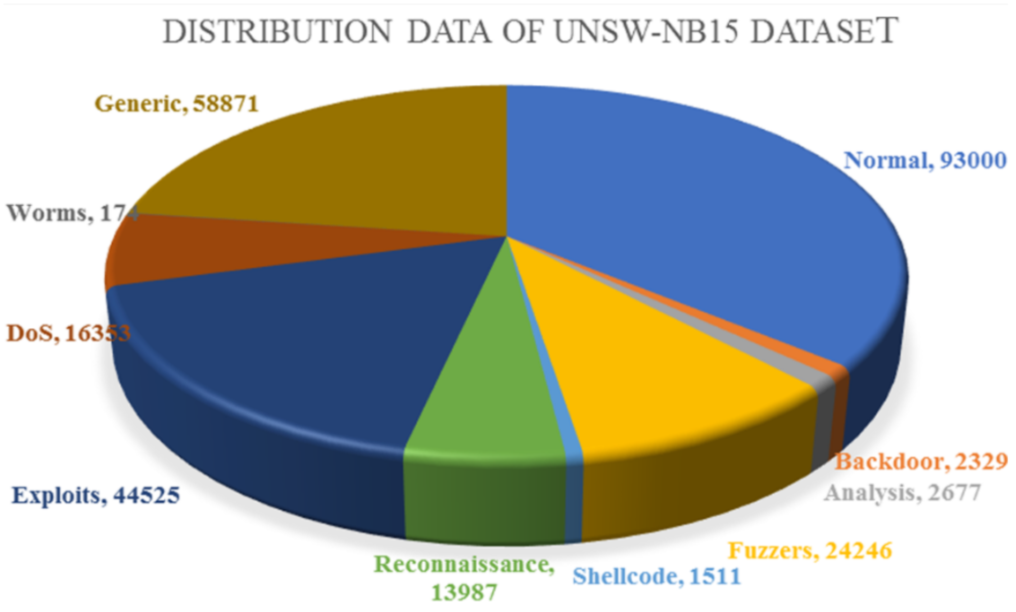


Figure 1.5: Distribution data of UNSW-NB15 dataset

Classes	**Training Subset**	**Testing Subset**
Normal	56,000	37,000
Analysis	2,000	677
Backdoor	1,746	583
DoS	12,264	4,089
Exploits	33,393	11,132
Fuzzers	18,184	6,062
Generic	40,000	18,871
Reconnaissance	10,491	3,496
Shellcode	1,133	378
Worms	130	44
Total Number of Records	**175,341**	**82,332**

Table 1.2: Number of records in training and testing subsets for each class

1.3.5 Challenges Addressed by UNSW-NB15

1.3.5.1 Modern Network Traffic Representation

The dataset includes recent network behaviors and protocols to provide a realistic simulation of traffic in modern networks.

1.3.5.2 Comprehensive Attack Types

UNSW-NB15 covers a wide range of attacks that are often seen in the real world, unlike older datasets that were limited in the types of attacks they included.

1.3.5.3 Balanced Dataset

By ensuring a balanced mix of both normal and malicious traffic, the dataset allows for more accurate training and testing of NIDS, improving the robustness of machine learning models.

1.3.5.4 Applications of the UNSW-NB15 Dataset

The UNSW-NB15 dataset is commonly used for the following purposes:
Machine Learning for Intrusion Detection: The dataset is highly utilized in developing machine learning algorithms for classifying network traffic as either normal or malicious.
Evaluating NIDS Performance: Researchers and organizations use the dataset to test and benchmark the performance of their intrusion detection systems.

Cybersecurity Research: It is widely applied in academic and industrial research to study new types of attacks and defenses.

1.3.5.5 Conclusion

The UNSW-NB15 dataset represents a significant advancement in the field of network intrusion detection by addressing the limitations of earlier datasets. By providing a comprehensive and diverse collection of modern network traffic, the dataset facilitates the effective training and evaluation of machine learning algorithms in classifying network behavior. With its rich feature set and representation of nine distinct attack types, it offers researchers and practitioners valuable insights into real-world cybersecurity threats. Moreover, the balanced composition of normal and malicious records ensures that machine learning models trained on this dataset can achieve higher accuracy and robustness. As cybersecurity threats continue to evolve, the UNSW-NB15 dataset remains a critical resource for ongoing research, development, and implementation of effective intrusion detection systems.

1.3.6 CICDDo2019 Dataset

1.3.6.1 Introduction

The Canadian Institute for Cybersecurity created the CICDDoS2019 dataset, which is an extensive tool for modelling and assessing Distributed Denial of Service (DDoS) assaults, especially when they pertain to Internet of Things devices. More than 2 million samples are included in this dataset, which includes both regular traffic and 15 different kinds of network-based assaults. Researchers creating detection and mitigation mechanisms for network security will find this dataset especially interesting because it focusses on volumetric assaults, which try to overload servers and networks with excessive traffic.

The CICDDoS2019 dataset, which was gathered under controlled laboratory conditions to mimic real-world circumstances, has 115 variables in total that are divided into characteristics related to network flow, transport layer, and application layer. Notably, it offers a comprehensive solution by addressing the increasing difficulty of identifying DDoS assaults directed at IoT devices, thus providing a well-rounded representation of network traffic. Pre-processing procedures were used to exclude characteristics that were strongly linked or unnecessary, as well as null and infinite values, in order to assure the quality of the data. In the end, this strong dataset helps improve intrusion detection systems specifically designed for

IOT networks by facilitating the creation and assessment of sophisticated machine learning models for identifying DDoS assaults in software-defined networking (SDN) settings.

1.3.6.2 Types of Attacks:

Here are the key types of attacks included in the CICDDoS2019 dataset:

- **UDP Flood**: Attackers send a massive number of UDP packets to random ports, overwhelming the target's resources and interrupting service.
- **TCP SYN Flood**: Exploits the TCP protocol's three-way handshake by sending SYN requests but never completing the connection, draining the target's resources.
- **HTTP Flood**: Overloads a web server with excessive HTTP requests, depleting its resources and bandwidth, leading to a denial of service.
- **ICMP Flood (Ping Flood)**: Attackers send a large number of ICMP Echo Request (ping) packets to the target, causing it to respond with Echo Replies, consuming processing power and bandwidth.
- **DNS Amplification**: The attacker sends spoofed DNS requests to legitimate DNS servers, which respond with larger payloads, saturating the target's bandwidth.
- **LDAP Amplification**: Attackers send small queries to LDAP servers, resulting in significantly larger responses that overwhelm the target system.
- **SNMP Amplification**: The attacker queries SNMP devices, causing them to respond with disproportionately large answers, overloading the victim's traffic.
- **NTP Amplification**: Attackers exploit NTP servers using the ”monlist” command to generate massive responses, similar to DNS amplification.
- **TFTP Amplification**: Small requests are sent to TFTP servers, which respond with much larger data packets, overwhelming the victim.
- **SYN-ACK Flood**: Attackers send SYN-ACK packets (part of the TCP handshake process) to the target, overwhelming its ability to handle network traffic.
- **UDP-Lag**: Exploits UDP services, especially in gaming or VoIP, by flooding the system with packets that cause lag or disruption.

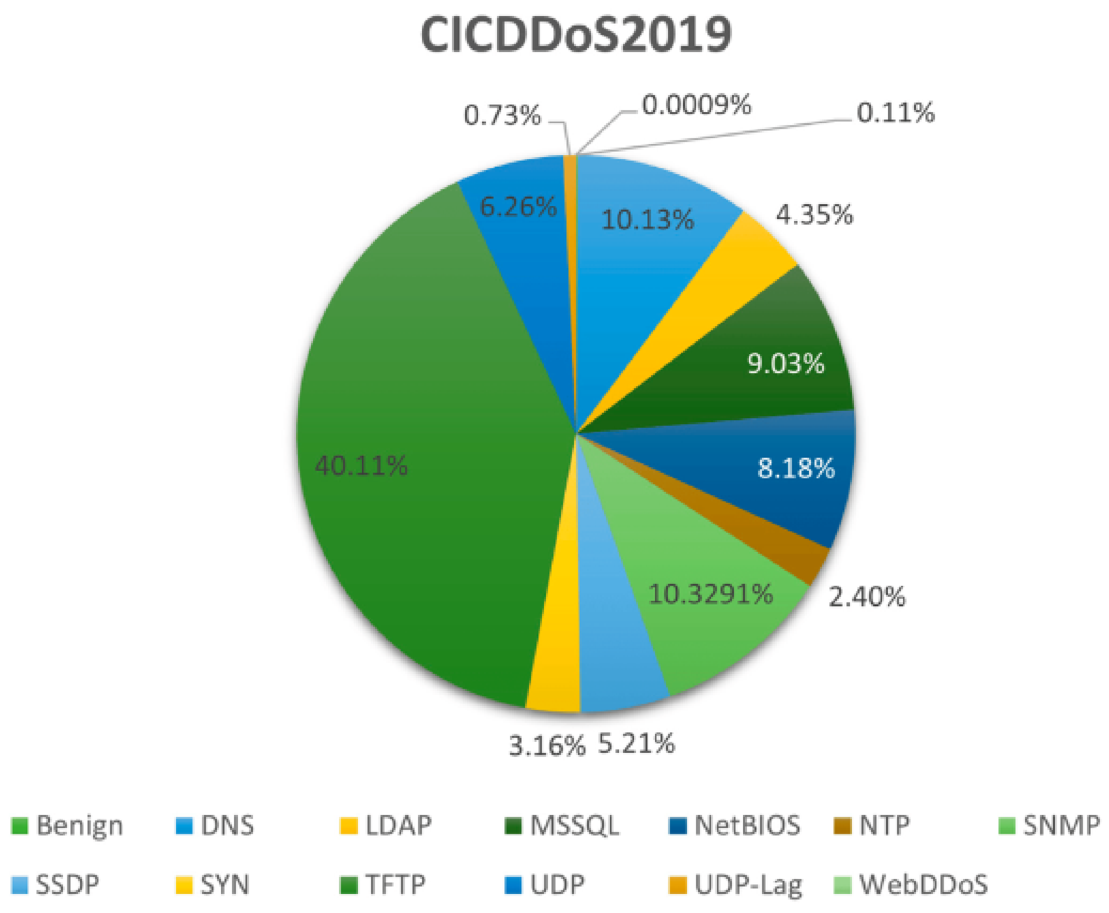


Figure 1.6: Distribution of traffic types in the CICDDoS2019 dataset

1.3.7 Features of the Dataset

The CICDDoS2019 dataset contains a rich set of features extracted from network traffic. These features capture various aspects of communication protocols and packet behavior.

1.3.7.1 Common Types of Features

- **Basic Flow Features**: Includes source and destination IPs, source and destination ports, protocols, and packet sizes.
- **Time-based Features**: Features related to the timing of packet arrivals, such as flow duration, inter-arrival times, and packet rates.
- **Content-based Features**: Features examining payload characteristics and protocol-specific behavior.

- **Traffic Features**: Metrics related to traffic volume, flow counts, and byte distribution over specific time windows.
- **Flag-based Features**: Information about TCP flags like SYN, ACK, FIN, etc.

The dataset typically contains **80+ features per record**, offering comprehensive insights into network traffic characteristics.

1.3.7.2 Number of Records

The dataset is extensive and includes millions of records. Each record represents a network flow. The data is collected over several days of attack simulations.

Training Data: Contains approximately 50 million records.
Testing Data: Also contains millions of records, with different distributions of attack and normal traffic.
Each record corresponds to a single network traffic flow or connection, making it suitable for machine learning algorithms focused on flow-based DDoS detection.

Attack Type	**Flow Count**
Benign	56,863
DDoS_DNS	5,071,011
DDoS_LDAP	2,179,930
DDoS_MSSQL	4,522,492
DDoS_NetBIOS	4,093,279
DDoS_NTP	1,202,642
DDoS_SNMP	5,159,870
DDoS_SSDP	2,610,611
DDoS_SYN	1,582,289
DDoS_TFTP	20,082,580
DDoS_UDP	3,134,645
DDoS_UDP-Lag	366,461
DDoS_WebDDoS	439

Table 1.3: Flow count for different attack types in the CICDDoS2019 dataset.

1.3.7.3 Labels

The dataset includes both normal and attack traffic, with labels that clearly distinguish between them. The primary labels are:

Normal: Refers to benign traffic.
Attack Types: These represent various DDoS attack categories simulated during the dataset collection.

The labels are provided in a binary or multiclass format:

- **Binary Classification**: Where the traffic is simply labeled as "normal" or "attack."
- **Multiclass Classification**: Where the traffic is labeled by specific attack types (e.g., UDP Flood, DNS Amplification, etc.).

1.3.7.4 File Structure

The dataset is typically organized into multiple CSV files, with each file representing a different attack scenario or day of collection. Each row in the CSV corresponds to one network flow, and columns correspond to the extracted features.

1.3.7.5 Label Distribution

The dataset features an imbalanced distribution of attack and normal traffic, reflecting real-world conditions. Researchers often need to handle class imbalance during model training through resampling techniques or performance metrics that accommodate imbalanced data.

1.3.7.6 Traffic Types

The dataset covers **both volumetric and application-layer DDoS attacks**. It provides a variety of scenarios to study DDoS behavior across different attack vectors and helps develop detection methods that work across multiple attack types.

1.3.7.7 Key Challenges Addressed

- **Realistic Simulations**: The dataset captures a wide range of real-world DDoS attack scenarios, reflecting different techniques and tactics used by attackers. This realism is crucial for researchers and practitioners, as it allows them to test and evaluate their detection and mitigation strategies under conditions that closely resemble actual network environments.
- **Class Imbalance**: DDoS datasets often suffer from class imbalance, with significantly more attack traffic than normal traffic. The CICDDoS2019 dataset includes a balanced mix, enabling researchers to develop algorithms that can effectively distinguish between

benign and malicious traffic. Addressing this imbalance is essential for training robust models that perform well in real-world applications.

- **Diversity in Attack Types**: The dataset encompasses various DDoS vectors, including both volumetric attacks (which aim to saturate bandwidth) and application-layer attacks (which target specific applications). This diversity supports the development of generalized models that can detect different types of attacks, enhancing the resilience of network defenses.
- **Volumetric vs. Application-Layer Attacks**: By including both types of attacks, the dataset allows researchers to analyze and compare detection methods specific to each category. This comprehensive analysis aids in understanding the nuances of how different attack types impact network performance and how they can be effectively mitigated.
- **Feature Engineering**: The rich set of features in the dataset provides a wealth of information for machine learning models. Effective feature engineering is critical for improving model accuracy, as it allows researchers to capture relevant patterns and behaviors associated with DDoS attacks. The dataset's variety of features supports detailed analysis and optimization of detection algorithms.
- **Scalability**: The dataset's large volume of records enables testing of detection algorithms on a significant scale. Researchers can evaluate the performance of their models under heavy traffic loads and ensure that their solutions can handle the demands of real-world environments, where high volumes of data are common.

1.3.7.8 Purpose and Applications

Detecting DDoS Attacks in Real-Time
Building Machine Learning Models for Anomaly Detection
Studying DDoS Attack Manifestations in Network Traffic
Developing Advanced Intrusion Detection Systems (IDS) and Classifiers

1.3.7.9 Conclusion

The CICDDoS2019 dataset represents an ultimate resource for researchers and developers in network security, providing a structured and detailed foundation for advancing DDoS detection and mitigation techniques. Its comprehensive feature set and realistic attack simulations facilitate significant advancements in defending against DDoS threats.

1.4 Conclusion

In conclusion, this project has provided a comprehensive exploration of network security, specifically focusing on the challenges posed by network anomalies and intrusions. Through an extensive literature review, we defined the fundamental concepts of network anomalies and intrusions, distinguishing between them while highlighting their causes and examples.

The investigation into various anomaly detection and intrusion detection systems revealed the critical role of machine learning, deep learning, and federated learning techniques in enhancing network security. These advanced approaches provide significant improvements in identifying and mitigating potential threats in real-time.

Additionally, we reviewed notable datasets such as UNSW-NB15 and CICDDo2019, which are instrumental in training and evaluating machine learning models for network intrusion detection. These datasets address modern network traffic patterns and diverse attack types, thus providing a robust foundation for research and development in this domain.

Chapter 2 Week2: Data understanding

Plan

2.1 Real Modern Network Activities vs. Synthetic Contemporary Attack Behaviors

Modern Network Activities and Synthetic Contemporary Attack Behaviors: An Overview

In today's increasingly interconnected world, recognizing the nature of normal network activities and distinguishing them from synthetic attack behaviors is crucial for digital security. Modern network activities refer to the everyday, legitimate operations that occur within a network, whereas synthetic attack behaviors are simulated hacks designed to resemble real-world hostile activities. These two concepts form the foundation of cybersecurity research and are essential for designing effective defenses against modern threats.

2.1.1 Modern Normal Activities in a Network Environment

Routine Network Activities: In any functioning network, various activities can be considered regular or legitimate. These operations ensure connectivity, the smooth flow of data, and provide essential services to users. Some of the most common legitimate network activities include:

- *User Authentication:* Routine login and logout requests are fundamental for network operation. These include authentication attempts using usernames and passwords, multi-factor authentication, and token-based systems. - *File Transfers:* Organizations often transfer valid files using protocols such as FTP, SFTP, or SMB. These transfers may involve data movement between internal servers, databases, or cloud storage services. - *Email Communication:* Email servers generate significant traffic when transmitting, receiving, and forwarding messages. Modern email systems use protocols such as SMTP, IMAP, and POP3, resulting in typical traffic patterns involving user interactions, spam filtering, and attachment transfers. - *Web Browsing:* Users accessing the internet via a network generate HTTP/HTTPS traffic when browsing websites, streaming content, or conducting online transactions. HTTPS encryption provides privacy and security. - *Application Access:* Employees often use internal and external applications, such as databases, enterprise resource planning (ERP) systems, and cloud-based services like Salesforce or Microsoft 365, generating specific patterns of API calls, database queries, and cloud traffic. - *Data Backup and Synchronization:* Many organizations automate data backups to local or cloud-based storage systems, generating predictable traffic patterns through services like Google Drive and

Dropbox. - *Network Management:* Administrators perform routine network management tasks, such as system updates, software patching, and hardware monitoring, which often involve pulling updates, logging system data, and setting up firewalls.

2.1.1.1 Characteristics of Legitimate Network Traffic

Legitimate network activities are often characterized by:

- *Predictable Timing:* Many network activities follow a set schedule, such as backups during off-hours or login spikes at the start of the workday. - *Expected Traffic Volume:* Routine operations generate consistent traffic over time, with occasional spikes during events like large file transfers. - *Secure and Encrypted Communications:* Most modern network operations use encryption (e.g., HTTPS, SSL, VPNs) to ensure data privacy, integrity, and security. - *Policy Compliance:* Legitimate network traffic adheres to organizational policies, such as role-based access control (RBAC) and secure file transfer protocols.

2.1.2 Synthetic Contemporary Attack Behaviors

As networks evolve, so do cyber threats. Synthetic attack behaviors are simulations of real-world cyberattacks that are carefully designed to mimic modern malicious techniques. These behaviors are commonly used to test, train, and evaluate cybersecurity systems on datasets or in experimental environments. The following sections describe various contemporary attack behaviors encountered in real-world scenarios and how they are emulated.

2.1.2.1 Types of Synthetic Attack Behaviors

1. **Phishing Attacks:** - *Simulation:* Phishing emails replicate real-world attacks, attempting to extract login credentials or install malware. Simulations often include fake credentials or links to sandboxed environments. - *Real-World Example:* Attackers send emails that appear to come from legitimate providers, tricking recipients into clicking malicious links or disclosing sensitive information.

2. **DoS and DDoS:** - *Simulation:* In synthetic environments, DDoS attacks are mimicked by generating massive amounts of traffic from multiple sources (bots) to overwhelm a target server or service. This may involve flooding HTTP requests or sending a large volume of ICMP packets.

3. **Ransomware Simulation:** - Simulated ransomware attacks encrypt data in a sandbox environment, allowing security teams to practice detection and recovery without

jeopardizing actual data. Simulation tools often create dummy files that are encrypted with test keys. - *Real-World Example:* A malicious payload is distributed via phishing emails or vulnerable machines, encrypting all available data and demanding a ransom for decryption keys.

4. **Man-in-the-Middle (MITM) Attacks:** - *Simulation:* MITM attacks are simulated by intercepting traffic between two parties (e.g., during a session handshake) and altering or stealing data. Tools like Ettercap or Wireshark are used to analyze the attack. - *Real-World Example:* Attackers intercept traffic between users and legitimate services, stealing login credentials or modifying messages in real-time.

5. **SQL Injection (SQLi):** - Attackers simulate SQL injection by submitting malicious queries to web applications with vulnerable input fields. This can be tested in sandboxed environments using test databases. - *Real-World Example:* An insecure web application allows attackers to inject SQL queries and retrieve unauthorized data from the backend database.

6. **Brute Force Attacks:** - *Simulation:* Brute force attacks simulate trying multiple password combinations on a login interface. These attempts can be throttled to match real-world attack rates. - *Real-World Example:* Attackers try various username and password combinations until they gain access to an account.

2.1.2.2 Characteristics of Synthetic Attack Traffic

Synthetic attack behaviors often resemble real attacks in terms of tactics and execution but are monitored in a lab or research setting. Common characteristics include:

- *Malicious Intent:* Unlike legitimate network activity, these behaviors aim to disrupt operations, steal data, or compromise system integrity. - *Abnormal Traffic Patterns:* Attack traffic is often characterized by unusual spikes, irregular patterns, or unexpected modes of communication, such as excessive login attempts or large data transfers at odd hours. - *Evasion Strategies:* Modern attack simulations include techniques to avoid detection, such as encrypting malicious payloads, using obfuscation, or launching attacks from geographically distributed locations (for DDoS). - *Adversarial Behavior:* Simulated attackers often try to bypass network defenses, such as firewalls or intrusion detection systems (IDS), by exploiting vulnerabilities.

2.1.2.3 Simulating Attack Behaviors for Research and Testing

In a research setting, attack behaviors are developed using penetration testing frameworks (e.g., Metasploit, Cobalt Strike) and datasets that reflect contemporary threats. These synthetic datasets are essential for training machine learning models, evaluating defense mechanisms, and raising overall cybersecurity awareness.

Datasets such as UNSW-NB15 and CICIDS 2017 contain labeled network traffic that simulates various attack vectors, including port scanning, brute force, and malware. These datasets allow cybersecurity professionals to distinguish between normal and abnormal traffic and train algorithms to detect emerging threats.

2.2 Tcpdump: A Crucial Tool for Network Capture and Analysis

What is tcpdump?

tcpdump is a crucial tool for capturing and analyzing networks. It allows for real-time traffic analysis or the recording of data for later analysis, with filters to select packets. Based on the `libpcap` library, it is compatible with other tools like Wireshark and works on multiple platforms. It is an essential tool for the administration and debugging of network applications.

How does tcpdump capture and analyze network traffic?

- **Packet Capture**
  - **Network Interface:** tcpdump can be run on a specific network interface (such as eth0, wlan0, etc.) to capture the packets flowing through that interface.
  - **Capture Filters:** You can specify filters to capture only the packets that meet certain criteria, such as a specific protocol (TCP, UDP, ICMP), a source or destination IP address, or a specific port.
- **Packet Analysis**
  - **Displaying Captured Packets:** By default, tcpdump displays summaries of the captured packets. You can use options to show more detailed information, including the packet headers.
  - **Protocol Decoding:** tcpdump supports several network and transport layer protocols, allowing it to decode relevant information within the captured packets, including IP addresses, ports, sequence numbers, and TCP flags.

- **Data Storage**
  - **Capture Files:** Captured packets can be saved in a pcap format file for later analysis with other tools, such as Wireshark.
- **Real-Time Analysis**
  - **Traffic Monitoring:** tcpdump allows real-time monitoring of network traffic, helping to identify issues, attacks, or abnormal behaviors.

2.2.1 Capture Steps with tcpdump for UNSW-NB15:

The tcpdump tool played a crucial role in capturing network packets, resulting in the generation of approximately 100 GB of raw data. Here's how this was specifically achieved:

1. **Network Configuration:** Researchers set up a laboratory network, integrating physical and virtual machines to simulate a modern environment with various services and protocols (HTTP, FTP, SSH, DNS, etc.). They generated both legitimate activities and different forms of attacks to create a mix of normal and malicious traffic.

2. **Using tcpdump for Capture:** tcpdump was used to capture all network traffic passing through one or more specific interfaces. The captures included packets from the transport and application layers, covering protocols such as TCP, UDP, ICMP, and other protocols used in both legitimate and malicious interactions.

3. **Application of Filters:** BPF (Berkeley Packet Filter) filters were applied to capture specific types of network traffic or to avoid capturing irrelevant packets, in order to reduce file size and improve the quality of the collected data.

4. **Data Volumes:** A total of 100 GB of raw data was captured over a given period, monitoring multiple services and network attacks. This significant amount resulted from capturing detailed packets, including headers and payload data.

5. **Generation of Attack Scenarios and Legitimate Traffic:** To make the dataset relevant for intrusion detection, researchers simulated various network attacks, such as denial-of-service (DoS) attacks, infiltrations, network scans, and other malicious techniques. Concurrently, legitimate traffic scenarios were executed to reflect normal operations within the network.

6. **Post-Processing of Captures:** After capturing data with tcpdump, post-processing was performed to extract important characteristics from each packet and packet aggregate (flow). Tools like Argus and Bro (Zeek) were likely used to analyze the .pcap files produced by tcpdump and to extract statistics such as IP addresses, ports, protocols, packet sizes, and connection durations.

7. **Construction of the Final Dataset:** The extracted network characteristics were then classified and integrated into the UNSW-NB15 dataset in the form of CSV or ARFF files, containing labels indicating whether the traffic was normal or malicious, as well as the type of attack.

2.2.2 Examples of Captured Attacks:

Among the types of attacks captured with tcpdump for UNSW-NB15, we find:

- **Reconnaissance (Scan):** Port scanning to detect open services.
- **DoS/DDoS:** Attacks aimed at making a service unavailable by flooding it with requests.
- **Injection of Malicious Packets:** Aimed at compromising services or hijacking sessions.

2.2.3 Storage of Captured Data

- **Pcap Format:** The packets captured by tcpdump are stored in files using the Pcap (Packet Capture) format. This format is a standard for recording network traffic and is widely used by various analysis tools such as Wireshark, Bro/Zeek, and Argus. A Pcap file contains the headers of network packets as well as the useful data, allowing for a comprehensive analysis of each data transmission.
- **Structure of Data in a Pcap File:** Each record in a Pcap file includes:
  - **Timestamp:** The moment the packet was captured.
  - **Packet Length:** The size of the captured packet.
  - **Packet Data:** The header of the packet (link layer, network, transport, etc.) and the useful data.

  - **Protocol Information:** The protocol used (IP, TCP, UDP, ICMP, etc.), source and destination IP addresses, port numbers, and other network information.

- **Volume and Segmentation of Pcap Files:** To capture 100 GB of raw data, the Pcap files were likely segmented into several smaller files for easier storage and analysis. Tcpdump allows for automatic segmentation of files based on size or duration, for example, by limiting each Pcap file to 1 GB.

- **Storage of Pcap Files:** The Pcap files created by tcpdump were stored on high-capacity disks. Given the size of the captured data (100 GB), high-performance storage systems were likely used to ensure that packet capture did not result in losses due to disk saturation.

2.3 Attack Types

The dataset incorporates nine types of attacks, which are reflective of real-world cybersecurity threats:

2.3.1 Fuzzers

Fuzzing is a method used to discover vulnerabilities in software or systems by sending a large volume of random, unexpected, or malformed inputs (data) to the target. The goal is to crash the system or expose weaknesses that could be exploited. Attackers use fuzzers to test the robustness of an application by bombarding it with inputs to trigger errors, buffer overflows, or other system malfunctions that can later be exploited.

Scenario example: In a real-world scenario, a fuzzer could target an e-commerce site by sending thousands of random, malformed inputs to vulnerable points, like login forms or search bars. For example, the attacker might use a fuzzing tool to overload the site's search bar with unusually long strings or special characters. If the website doesn't handle these inputs properly, it could crash or reveal weaknesses, such as SQL injection vulnerabilities or buffer overflow issues. The attacker could then exploit these bugs to gain unauthorized access, steal sensitive data, or disrupt services, resulting in financial loss and reputational damage for the business.

2.3.2 Analysis

Analysis attacks focus on reconnaissance activities, where attackers attempt to gather detailed information about the target network. This can include port scanning (to find open ports and services), spam activities, or other probing techniques to map the network and identify vulnerabilities for future exploitation.

Scenario example: In a real-world analysis attack scenario, an attacker could perform a port scan on a company's server to gather information about open ports and services running on the system. For example, the attacker might use tools like Nmap to scan the server, identifying services like HTTP (port 80) that are open and potentially vulnerable. With this information, they can plan more targeted attacks, such as attempting to exploit outdated software versions or weak configurations, leading to unauthorized access or data breaches.

2.3.3 DoS (Denial of Service)

A DoS attack aims to overwhelm a target system, server, or network with an excessive amount of traffic or requests, causing it to become unavailable to legitimate users. This overload can exhaust resources such as CPU, memory, or bandwidth, leading to service disruptions or complete outages.

Scenario example: On launch day, the popular gaming platform "GameZone" faces overwhelming excitement as thousands of players log in for a highly anticipated game. Suddenly, a DDoS attack hits, with a botnet flooding the servers with fake traffic, causing severe lag and disconnects for legitimate users. The IT team scrambles to mitigate the attack using various defenses, but restoring normal operations proves challenging. As frustrated players take to social media, the company grapples with negative publicity and the need to rebuild trust with its community.

2.3.4 Exploits

An exploit attack takes advantage of vulnerabilities in software or systems to gain unauthorized access or perform malicious actions. Attackers identify security flaws, such as unpatched software or coding errors, and use specific techniques to manipulate these weaknesses. Successful exploits can lead to data breaches, system outages, or unauthorized control, resulting in significant damage.

Scenario example: A popular social media platform experienced an exploit attack when attackers discovered a flaw in its API. By leveraging this vulnerability, they accessed private user messages and account information for millions of users. The breach prompted an investigation and significant media coverage, resulting in a loss of user trust and calls for stronger security measures.

2.3.5 Generic

A generic type of attack refers to a broad category of cyber threats that target various systems using common tactics, such as phishing, social engineering, denial-of-service, or brute-force attacks. These attacks often require minimal technical skills, making them accessible to a wide range of attackers.

Scenario example: An employee at a company receives a seemingly legitimate email that appears to be from their IT department, requesting them to verify their account credentials through a provided link. Trusting the source, the employee clicks the link and enters their username and password on a fake login page. The attackers capture this information and gain unauthorized access to the company's network.

2.3.6 Reconnaissance

A reconnaissance attack is the initial phase of a cyber attack where the attacker gathers information about a target to identify vulnerabilities and plan their approach. This phase can involve techniques such as network scanning, social engineering, and footprinting to collect data on IP addresses, open ports, and employee details.

Scenario example: An attacker targets a mid-sized company and begins a reconnaissance phase by conducting online searches for employee information, such as job titles and contact details. They utilize tools to scan the company's website and identify the server's IP addresses and open ports. The attacker also checks social media platforms to gather insights about the company's software and security practices.

2.3.7 Shellcode

A shellcode attack involves injecting malicious code into a target system by exploiting vulnerabilities, such as buffer overflows, in software applications. Attackers craft a small piece of code, known as shellcode, designed to execute specific commands, like opening a command shell or installing malware.

Scenario example: An attacker discovers a vulnerability in a widely used web application that suffers from a buffer overflow issue. They craft a malicious payload containing shellcode designed to open a command shell on the server. By exploiting the vulnerability, the attacker sends a specially crafted input that overwrites the application's memory, injecting their shellcode.

2.3.8 Worms

Worms are a type of malware that replicate themselves to spread across networks and systems without the need for human intervention. Unlike viruses, which attach themselves to files or programs, worms are standalone software that exploit vulnerabilities in operating systems or applications to propagate.

Scenario example: A large corporation's network is infected by a worm that exploits a vulnerability in its outdated software. Once the worm infiltrates one computer, it begins to replicate itself, scanning for other connected devices on the network.

2.3.9 Backdoors

A backdoor attack involves the installation of hidden methods for bypassing normal authentication processes to gain unauthorized access to a system or network. Attackers often exploit vulnerabilities or use malware to create a backdoor, allowing them to control the system remotely without detection.

Scenario example: An attacker gains access to a corporate network by exploiting a vulnerability in a web application. After successfully breaching the system, they install a backdoor that allows them to bypass normal security protocols.

2.4 Argus and Bro-IDS

Argus and Bro-IDS are powerful network intrusion detection systems (NIDS) that play a crucial role in the UNSW-NB15 dataset. They are used to capture network traffic data and extract relevant features for intrusion detection.

2.4.1 Argus

Argus carries out the following crucial actions in order to function as a network traffic analyzer:

1. **Packet Capture**: Network Interface Monitoring: Different network interfaces, such as Ethernet and Wi-Fi, are configured in Argus to be monitored. All network packets traveling across these interfaces are intercepted by it.

2. **Protocol Decoding**:

   - Packet Parsing: To determine the network protocol (such as TCP, UDP, ICMP, or HTTP), Argus examines the packet headers.
   - Extracting Data: This includes IP addresses, ports, protocols (ICMP, UDP, TCP), payload, and timestamps.

3. **Traffic Analysis**: Argus computes various statistics like packet counts, byte counts, and inter-arrival times. It can also detect anomalies in the traffic patterns and generate alerts.

4. **Data Presentation**: Argus provides visualization capabilities like packet flow diagrams, traffic charts, and protocol distribution.

2.4.2 Bro-IDS (Zeek)

Bro-IDS, now known as Zeek, is another advanced network intrusion detection system that uses both signature-based and anomaly-based detection. It is designed for deep packet inspection and is highly extensible.

1. **Packet Capture**: Similar to Argus, Bro-IDS captures network packets from different interfaces and stores them temporarily for further processing.

2. **Protocol Analysis**:

   - Parsing Network Protocols: Bro-IDS analyzes headers from all network layers (link, network, transport, and application) and inspects the payload of packets.
   - Event Generation: Bro-IDS operates based on event-driven logic, transforming network traffic into high-level events like login attempts or file downloads.

3. **Script-Based Analysis**: Users can write custom scripts for specific network behaviors, making Bro-IDS highly customizable.

4. **Anomaly Detection**: Bro-IDS uses both signatures for known threats and behavioral analysis for zero-day attacks.

5. **Alerting and Logging**: Bro-IDS generates real-time alerts for suspicious activity and keeps extensive logs for forensic analysis.

2.4.3 12 Algorithms

1. **Network Header Extraction Algorithm**

   - **Purpose:** Extracts data from the headers of network packets, which contain essential metadata about the transmission, such as source/destination IP addresses, protocol types, and ports.
   - **Features Generated:** IP addresses, port numbers, protocol type, packet size, flags, and time-to-live (TTL).
   - **Significance:** By extracting header information, it enables network monitoring, packet filtering, traffic classification, and the identification of abnormal traffic patterns.

2. **Flow Analysis Algorithm**

   - **Purpose:** Analyzes network traffic by grouping packets into flows, which are sequences of packets sharing common properties (e.g., IP address, protocol).
   - **Features Generated:** Flow duration, packet count, byte count, flow start/end times, and flow direction.
   - **Significance:** Helps in identifying traffic trends, application behaviors, and anomalies in communication patterns, which is critical for network optimization and security monitoring.

3. **Time Statistics Analysis Algorithm**

   - **Purpose:** Focuses on analyzing the temporal characteristics of network traffic, like packet arrival times and intervals between packets.
   - **Features Generated:** Packet inter-arrival times, flow duration, time of day activity, and jitter.
   - **Significance:** Key in detecting time-based anomalies, such as denial-of-service attacks, and understanding network performance (e.g., latency, throughput).

4. **Protocol Analysis Algorithm**

- **Purpose:** Analyzes the types of network protocols being used (e.g., TCP, UDP, ICMP) to identify trends and detect misuse of certain protocols.
- **Features Generated:** Protocol distribution, protocol types per flow, protocol-specific flags (e.g., SYN/ACK in TCP).
- **Significance:** Enables protocol-based filtering, detection of protocol misuse (e.g., tunneling attacks), and optimization of protocol use within a network.

5. **Connection State Detection Algorithm**
   - **Purpose:** Monitors and determines the state of network connections, identifying whether connections are active, closed, or pending.
   - **Features Generated:** Connection states (e.g., open, closed, SYN-sent), session initiation/termination times.
   - **Significance:** Crucial for tracking the lifecycle of network sessions, identifying unauthorized connection attempts, and managing resources efficiently.
6. **Network Load Calculation Algorithm**
   - **Purpose:** Measures and analyzes the overall network load to evaluate performance and detect potential bottlenecks or overload conditions.
   - **Features Generated:** Bandwidth usage, packet and byte rates, load distribution across network segments.
   - **Significance:** Helps in network capacity planning, identifying overloaded resources, and detecting traffic surges indicative of attacks like DDoS.
7. **Anomaly Detection Algorithm**
   - **Purpose:** Identifies deviations from established baseline traffic patterns, indicating potential security threats or misconfigurations.
   - **Features Generated:** Anomalous traffic flags, outlier detection, baseline deviation metrics.
   - **Significance:** Essential for detecting zero-day attacks, unusual traffic patterns, and early warning signs of intrusions or misconfigurations.
8. **Statistical Aggregation Algorithm**

- **Purpose:** Aggregates various statistical metrics across network data, such as packet counts, flow metrics, and protocol distribution.
- **Features Generated:** Summarized traffic metrics (e.g., total packets, total bytes, average flow size).
- **Significance:** Facilitates trend analysis, network performance evaluations, and the detection of network-wide anomalies.

9. **Session Analysis Algorithm**
    - **Purpose:** Analyzes the communication sessions between two or more devices, tracking their behavior, timing, and duration.
    - **Features Generated:** Session duration, session byte count, session frequency.
    - **Significance:** Helps in understanding user and application behavior, detecting session hijacking, and monitoring long-duration or suspicious sessions.

10. **Security Indicator Calculation Algorithm**
    - **Purpose:** Calculates specific security-related metrics that indicate the health and security status of a network, based on network activity and threat intelligence.
    - **Features Generated:** Risk scores, threat indicators, intrusion likelihood, security event frequency.
    - **Significance:** Provides actionable insights for security teams, helping prioritize incident responses and mitigate risks effectively.

11. **Connection Frequency Analysis Algorithm**
    - **Purpose:** Analyzes the frequency of connections between hosts or services, identifying unusual or overly frequent connection attempts.
    - **Features Generated:** Connection counts, average connection frequency, deviation from normal connection rates.
    - **Significance:** Useful for detecting scanning activities, brute-force attempts, and identifying machines participating in botnets or DDoS attacks.

12. **Attack Pattern Identification Algorithm**
    - **Purpose:** Detects patterns in network traffic that resemble known attack signatures or behaviors, aiding in identifying ongoing or potential attacks.

- **Features Generated:** Attack signatures, matched patterns, anomaly scores, attack likelihood.
- **Significance:** Critical for real-time threat detection, such as identifying SQL injection, cross-site scripting, or denial-of-service attacks based on traffic behavior.

2.5 Ground Truth Dataset

2.5.1 Definition

A ground truth dataset is a meticulously labeled collection of data that serves as a benchmark for testing how well your machine learning models perform. Think of it as a reliable reference book that guides you in training and validating models, ensuring they make accurate predictions or classifications.

It's important to note that ground truth datasets aren't just any old data repository. Instead, they contain expert topic knowledge, contextual understanding, and detailed insights needed to build advanced AI models that solve real problems effectively.

Unlike generic datasets, every label or annotation in a ground truth dataset is carefully verified for accuracy. This level of scrutiny ensures your AI models are built on solid foundations, incorporating domain expertise and nuanced insights to tackle real-world challenges effectively.

2.5.2 Contextual Information

The ground truth table associates a label with each record in a dataset (e.g., a network packet), indicating its true nature:

- **Benign:** The traffic is normal and poses no threat.
- **Malicious:** The traffic is associated with an attack or suspicious activity.

This label, provided by security experts, allows you to compare the results obtained by a model with reality.

2.5.3 Structure and Use for Evaluating Model Performance

A ground truth table is typically structured as a CSV (Comma Separated Values) file or similar. It contains at least two columns:

- **Record ID:** A unique identifier for each record in the dataset.
- **Label:** Indicates whether the record is benign or malicious.

2.5.4 Using the Ground Truth Table to Evaluate a Model

To evaluate a model using this table, follow these steps:

1. **Prediction:** Pass the dataset through the model. For each record, the model produces a prediction (benign or malicious).
2. **Comparison:** Compare the model's predictions with the labels from the ground truth table.
3. **Calculate Metrics:** Calculate evaluation metrics such as:
   - **True Positive Rate (TPR):** Proportion of attacks correctly detected.
   - **False Positive Rate (FPR):** Proportion of benign events incorrectly classified as attacks.
   - **True Negative Rate (TNR):** Proportion of benign events correctly classified.
   - **False Negative Rate (FNR):** Proportion of attacks incorrectly classified as benign.
   - **Precision:** Proportion of positive instances correctly identified out of all instances identified as positive.
   - **Recall:** Proportion of positive instances correctly identified out of all actual positive instances.
   - **F1-score:** Harmonic mean of precision and recall.

In summary, the `UNSW-NB15_GT.csv` ground truth table serves as a reference for evaluating a model's ability to distinguish attacks from normal behavior in a specific dataset. By comparing the model's predictions with the actual labels, you can quantify the model's performance and identify its strengths and weaknesses.

2.6 Role of the Event List File in the UNSW-NB15 Dataset

2.6.1 Introduction

The UNSW-NB15 dataset is a popular benchmark for network intrusion detection systems (NIDS). The Event List File (`UNSW-NB15_LIST_EVENTS.csv`) is an essential component that helps track and categorize events that occur during the data collection process. This file contains rich metadata about network traffic events and their associated attack labels, helping researchers and analysts to better understand the context of each captured network flow.

2.6.2 Purpose of the Event List File

The event list file in the UNSW-NB15 dataset serves the following critical functions:

1. **Monitoring Network Events** The primary goal of the event list file is to provide a thorough log of all significant events that occurred during data collection. Each row in the file corresponds to a specific event, which includes important details such as the event type, duration, and other parameters that can be used to characterize network traffic.

   By recording these events, the event list file creates a structured picture of network activity timelines and identifies patterns or abnormalities that could indicate potential security breaches.

2. **Classifying Events by Type** The Event List File categorizes network events into several categories, which often include:

   - **Normal Events:** Routine, lawful network traffic that conforms to typical user behaviors and system operations.
   - **Attack Events:** Malicious behavior intended to compromise the system, steal data, or disrupt operations. The file comprises a variety of attack types, including denial-of-service (DoS), backdoors, and worms.

   This classification is essential for distinguishing between benign and malicious communication, as it is used to train and assess intrusion detection systems.

3. **Providing Event Metadata** Each event is tagged with a number of information attributes that assist researchers in understanding the activity's context and nature. These fields can include:

   - **Event ID:** Each event is assigned a unique identification.
   - **Timestamp:** The precise time that the event occurred.
   - **Source and Destination IP/Ports:** The IP addresses and ports used throughout the conversation.
   - **Occurrence Type:** Whether the occurrence is typical or indicates a specific type of attack.
   - **Attack Label:** Detailed labels that identify the sort of attack, such as "SQL Injection," "Port Scan," or "Brute Force."

2.6.3 Role in Network Intrusion Detection

The Event List File is essential for training and testing network intrusion detection systems (NIDS) on the UNSW-NB15 dataset. It provides the ground truth for machine learning models and statistical tools to distinguish between normal and harmful behavior. By providing thorough descriptions of each occurrence, the file allows models to:

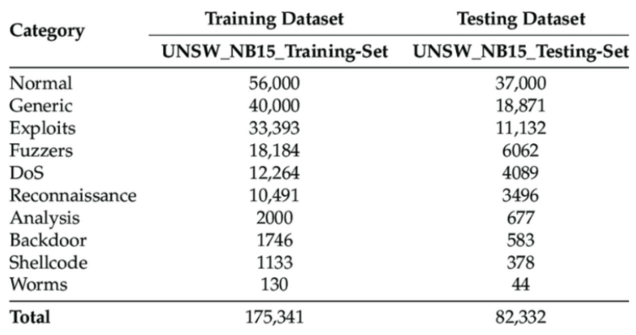
- **Detect Intrusions:** By learning from labeled events, NIDS can identify new instances of similar attacks in live environments.
- **Improve Model Precision:** The granularity of event metadata allows models to be fine-tuned, minimizing the likelihood of false alarms by precisely discriminating between genuine traffic and attack routes.

2.7 Dataset Partitioning and Proposals

2.7.1 1.1 Dataset Partitioning

The partition of the full dataset is divided into a training set and a test set according to the hierarchical sampling method, namely, `UNSW-NB15-training-set.csv` and `UNSW-NB15-testing-set.csv`. The training dataset consists of 175,341 records, whereas the testing dataset contains 82,332 records. The number of features in the partitioned dataset [11] is different from the number of features in the full dataset [8]. The partitioned dataset has only 43 features with the class label, removing 6 features (i.e., `dstip`, `srcip`, `sport`, `dsport`, `Ltime`, and `Stime`) from the full dataset. The partitioned dataset contains ten categories, one normal and nine attacks, namely, generic, exploits, fuzzers, DoS, reconnaissance, analysis, backdoor, shellcode, and worms.

This table shows in detail the class distribution of the UNSW-NB15 dataset.



Category	Training Dataset	Testing Dataset
	UNSW_NB15_Training-Set	UNSW_NB15_Testing-Set
Normal	56,000	37,000
Generic	40,000	18,871
Exploits	33,393	11,132
Fuzzers	18,184	6062
DoS	12,264	4089
Reconnaissance	10,491	3496
Analysis	2000	677
Backdoor	1746	583
Shellcode	1133	378
Worms	130	44
Total	175,341	82,332

Figure 2.1: Class distribution of the UNSW-NB15 dataset

2.7.2 1.2 Rationale Behind This Partition

Training Ensemble (70%): This ensemble is necessary for the model to learn the relationships between the characteristics and the output classes. Enhancing the robustness of the model is possible with a large number of examples.

Test Ensemble (30%): This ensemble serves to evaluate the final performance of the model. By keeping this distinct data, one can have a realistic estimate of the model's performance in an actual scenario.

2.7.3 1.3 Alternative Strategies for Partitioning Suggestions

2.7.3.1 a. Including a Validation Ensemble

It would be beneficial to include a separate validation group in the partitioning procedure. This group makes it possible to modify the model's hyperparameters without affecting the final assessment. One typical procedure could be as follows:

- 60% of the training group
- Validation ensemble: 20%
- Test ensemble: 20%

2.7.3.2 b. The Significance of a Validation Ensemble

- **Modification of the Hyperparameters:** The validation process as a whole enables testing several model configurations (such as algorithm selection, feature selection, etc.) to determine which one provides the best performance.

- **Avoidance of Overfitting:** By using a validation ensemble, one can determine whether the model starts to overadjust training data by monitoring performance on validation data.

2.7.4 1.4 Alternative Partitioning Strategies

Here are a few other tactics that could be considered:

- **K-fold Cross-Validation:** Split the dataset into K sub-ensembles, then train and test the model K times, using a different sub-ensemble as the test ensemble and the remaining sub-ensemble as the training ensemble each time. This makes it possible to minimize the variance in performance estimates and optimize the use of data.

- **Stratified Sampling:** Ensuring that each partition (training, validation, test) contains comparable proportions of each class is crucial, especially if some classes are underrepresented.

- **Temporal Partitioning:** It may be wise to divide training and testing groups according to time if the data include a temporal component (for example, use training data from the first month and testing data from the subsequent month).

2.7.5 1.5 In Summary

A crucial step in the creation of a machine learning model is the partitioning of datasets. With recommended percentages of 60% for training, 20% for validation, and 20% for testing, a conventional partition for the UNSW-NB15 dataset might include training, validation, and test ensembles. These decisions ensure that the model can be formed and evaluated effectively while minimizing overfitting risks and optimizing performance.

2.8 Exploratory Data Analysis Report on the UNSW-NB15 Dataset

2.8.1 Introduction

This section outlines the data preprocessing and exploratory data analysis (EDA) conducted on the UNSW-NB15 dataset. The primary goal is to clean, analyze, and prepare the dataset for machine learning model building. The dataset consists of network traffic data, where each record represents a network connection with multiple features such as packet counts, byte counts, and connection states. The target variable, `attack_cat`, categorizes each connection into one of several attack types or "normal" behavior.

2.8.1.1 Key Objectives

The following objectives were targeted in the analysis:

- Inspect the dataset and understand its structure.
- Handle missing data and duplicate records.
- Explore the statistical characteristics of the dataset.
- Detect and address outliers.
- Visualize the distribution of key features.
- Perform feature engineering to enhance model performance.
- Address class imbalance using techniques like SMOTE.

2.8.2 Data Loading and Initial Inspection

2.8.2.1 Libraries Used

The following Python libraries were imported for data manipulation, visualization, and analysis:

- **Pandas**: for data manipulation and DataFrame analysis.
- **Numpy**: for numerical computations.
- **Matplotlib** and **Seaborn**: for data visualization.

- **Scikit-learn**: for machine learning techniques such as PCA and K-Means.

2.8.2.2 Data Loading

The training and testing datasets were loaded from CSV files stored on Google Drive. The initial inspection of the dataset revealed the following details about the training data:

- **Total rows and columns**: 45 columns and 175,341 entries.
- **Data types**: Various numeric and categorical columns.

2.8.2.3 Data Structure

The dataset contains several columns, both numeric and categorical. The initial inspection revealed:

- 30 numeric columns (types: `int64` and `float64`).
- 4 categorical columns (type: `object`).

2.8.2.4 Descriptive Statistics

The dataset consists of 175,341 entries and 45 columns. The key statistics include:

- **Numeric columns**:
  - `Duration (dur)`: Mean = 1.36, Std Dev = 6.48, Min = 0.00, Max = 59.99.
  - `Number of packets sent (spkts)`: Mean = 20.30, Max = 9616.
  - `Bytes sent (sbytes)`: Mean = 8844.84, Max = 12,965,230.
- **Categorical column**:
  - `attack_cat`: Most frequent category is `Normal`, with 56,000 occurrences.

2.8.2.5 Concatenating the Data

The multiple CSV files (NB15_1, NB15_2, NB15_3, NB15_4) containing the data were combined into a single training dataset using `pd.concat()`. The column names were standardized using the feature names provided in `NB15_features.csv`. The following steps were performed:

- Renamed columns based on feature names.

- Combined multiple datasets into a single `train_df` dataset.
- Shuffled the data to ensure randomness in row order.

2.8.3 Data Preprocessing

2.8.3.1 Handling Duplicates

We checked for duplicate records using `train_df.duplicated()` and removed any duplicates to ensure data integrity. This avoided potential bias or overrepresentation of certain records.

2.8.3.2 Handling Missing Values

Missing values were analyzed, and strategies were applied to handle them:

- `attack_cat`: Missing values were replaced with `normal`.
- `ct_flw_http_mthd`: Missing values were replaced with 0 (no HTTP method).
- `is_ftp_login`: Missing values were replaced with 0 (no FTP login).

A summary table before and after handling missing values was produced to evaluate the effectiveness of the approach.

2.8.3.3 Descriptive Analysis

We performed a statistical summary of both numeric and categorical features:

- **Numeric features**: Analyzed using `train_df.describe()` to understand distributions, means, standard deviations, and ranges.
- **Categorical features**: Analyzed using `train_df.describe(include="O")`, which provided counts and distributions for categorical features.

2.8.4 Exploratory Data Analysis (EDA)

2.8.4.1 Outlier Detection and Handling

Outliers can significantly impact model performance, especially in algorithms sensitive to extreme values. We identified outliers using the Interquartile Range (IQR) method and visualized their distribution through boxplots. Outliers were handled by replacing values outside 1.5 * IQR with the median.

2.8.4.2 Class Distribution

The distribution of attack categories was visualized using bar graphs to understand the proportion of each class. Additionally, pie charts were used to further illustrate the class distribution.

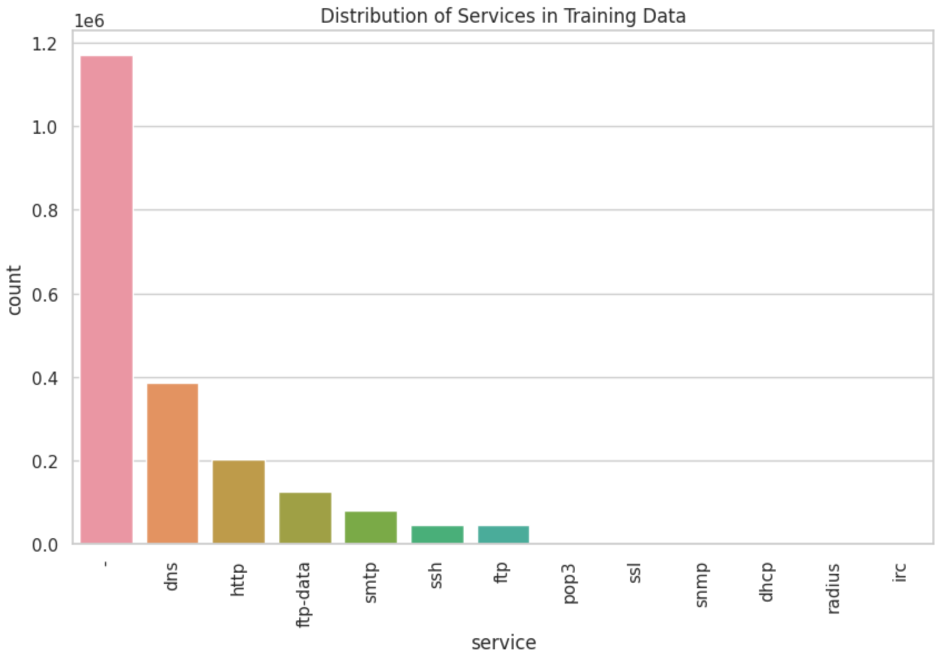


Figure 2.2: Distrubition des services dans training Dataset

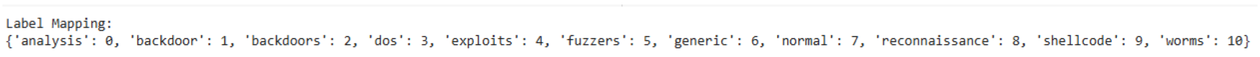
2.8.4.3 Feature Engineering

We created new features to enrich the dataset and improve model performance:

- **Ratios**: `byte_ratio` (source bytes/destination bytes), `pkt_ratio` (source packets/destination packets).
- **Aggregations**: `total_bytes` (sum of source and destination bytes), `total_pkts` (sum of source and destination packets).

2.8.4.4 Target Feature Encoding

The target feature `attack_cat` was encoded numerically using `LabelEncoder`, which converts categorical attack classes into numerical values.



```
Label Mapping:
{'analysis': 0, 'backdoor': 1, 'backdoors': 2, 'dos': 3, 'exploits': 4, 'fuzzers': 5, 'generic': 6, 'normal': 7, 'reconnaissance': 8, 'shellcode': 9, 'worms': 10}
```

Figure 2.3: Target Feature encoding

2.8.4.5 Correlation Analysis

A correlation matrix was computed to identify highly correlated features. Features with correlations above 0.75 were flagged, and one of each highly correlated pair was dropped to avoid multicollinearity.

2.8.4.6 Dimensionality Reduction

Principal Component Analysis (PCA) was used to reduce the dimensionality of the dataset while retaining as much variance as possible. The cumulative explained variance was plotted, and the optimal number of components (e.g., 10 components capturing over 90

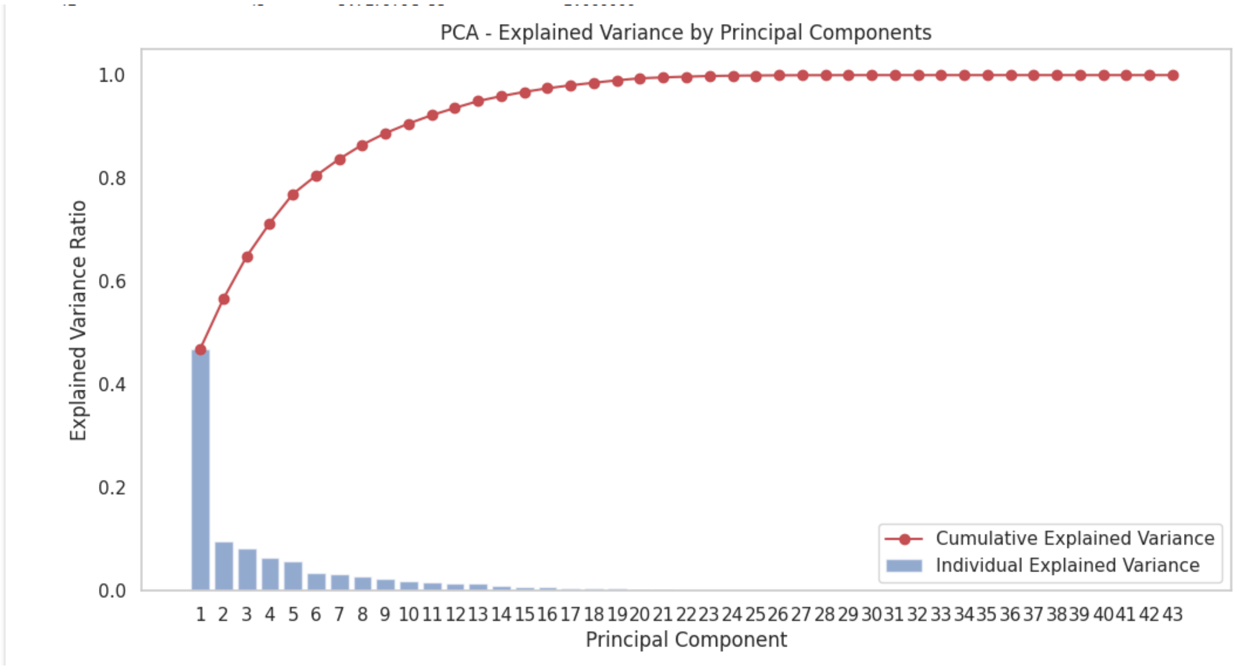


Figure 2.4: Applying PCA

2.8.5 Class Imbalance Handling

The dataset suffers from class imbalance, with some attack categories being underrepresented. To address this issue:

- **SMOTE (Synthetic Minority Over-sampling Technique)**: Applied to the minority classes to generate synthetic examples.
- **Random Under-Sampling**: Used to reduce the size of the majority classes and prevent overrepresentation.

A comparison of class distribution before and after applying SMOTE and under-sampling was provided.

2.8.6 Feature Importance Analysis

Random Forest models were used to assess the importance of features based on their contribution to reducing impurity in decision trees. Features with an importance score higher than 0.005 were retained, and those with lower importance were eliminated.

2.8.7 Conclusion

The exploratory analysis of the UNSW-NB15 dataset has significantly improved our understanding of the data. We have successfully handled missing values, outliers, and class imbalance, while performing feature engineering and dimensionality reduction. The next steps will involve training machine learning models for intrusion detection, tuning hyperparameters, and evaluating their performance on the test dataset.

Chapter 3

Week 3: Data Preparation

Plan

3.1 Introduction

This section covers the preparation of the UNSW-NB15 dataset's data for modeling. Thorough preparation is necessary to ensure that the data are accurate, relevant, and ready for use in machine learning models. The steps include data cleaning, feature engineering, normalization, and class imbalance management.

3.2 Data Cleaning

3.2.1 Handling Missing Values

3.2.1.1 Identification of Missing Values

There are 2,218,760 missing values in the dataset overall, mostly in the `attack_cat` column. Since a model cannot handle missing values well, this constitutes a sizable fraction of the data, which can present difficulties during modeling. Since the attack category is crucial for intrusion classification, the `attack_cat` column is very significant.

3.2.1.2 Imputation of Missing Values

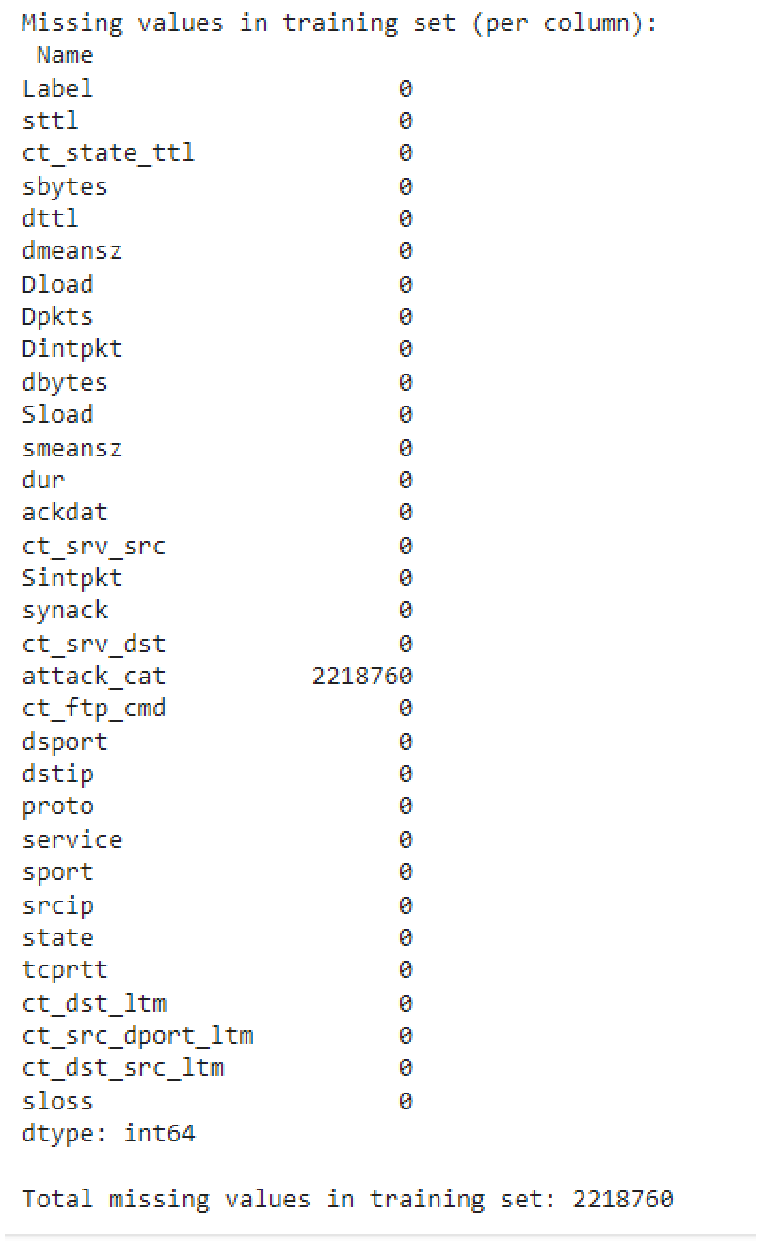
We applied imputation techniques on these missing values:

- **Imputation of Numerical Columns:** Mean imputation is the approach selected. Because it substitutes the mean value of the other data in the same column for missing values, this approach works well with numerical data. This method fills in the gaps while maintaining the data's distribution.
- **Imputation of Categorical Columns:** We employed the most frequent value for imputation in categorical columns. This approach substitutes the most often occurring category for missing values. By doing this, bias is prevented from entering the data and the categories' integrity is preserved.

After imputation, the results show that all columns now contain no missing values. This means that the dataset is now complete and ready for further analysis and modeling.

3.2.2 Outlier Management

Outlier management is a crucial step in data processing, as outlier values can significantly influence the results of statistical analyses and machine learning models. In our study, we



```
Missing values in training set (per column):
 Name
Label                     0
sttl                      0
ct_state_ttl              0
sbytes                    0
dttl                      0
dmeansz                   0
Dload                     0
Dpkts                     0
Dintpkt                   0
dbytes                    0
Sload                     0
smeansz                   0
dur                       0
ackdat                    0
ct_srv_src                0
Sintpkt                   0
synack                    0
ct_srv_dst                0
attack_cat          2218760
ct_ftp_cmd                0
dsport                    0
dstip                     0
proto                     0
service                   0
sport                     0
srcip                     0
state                     0
tcprtt                    0
ct_dst_ltm                0
ct_src_dport_ltm          0
ct_dst_src_ltm            0
sloss                     0
dtype: int64

Total missing values in training set: 2218760
```

Figure 3.1: Missing Values in the Dataset

used the Interquartile Range (IQR) method to detect and handle outliers in the dataset.

3.2.2.1 Outlier Detection Using the IQR Technique

The interquartile range (IQR) is a measure of statistical dispersion that describes the range within which the central 50% of the data lies. Here are the steps for detecting outliers:

- **Quartile Calculation:**

  - The value that 25% of the data falls below is denoted as Q1 (First Quartile).
  - The third quartile, or Q3, is the value that 75% of the data falls below.
  - The formula for calculating IQR is IQR = Q3 Q1.

- **Setting Boundaries:**
  - Lower Bound: $Q1 - 1.5 \times IQR$
  - Upper Bound: $Q3 + 1.5 \times IQR$

  Outliers are values that deviate from these bounds.

3.2.2.2 Handling Outliers

To eliminate outliers from each of the dataset's numerical columns, we used the `remove_outliers_iqr` function. Following this process, the dataset's structure changed to (463,707, 32), meaning that 32 features and 463,707 samples are still present.

3.2.2.3 Visualization of Results

We also generated a boxplot to visualize the distribution of the `sbytes` and `dbytes` features after removing the outliers.

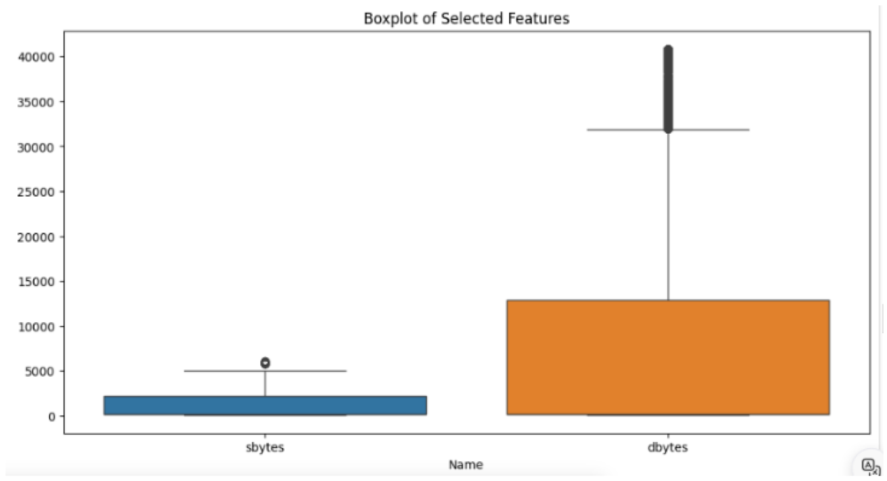


Figure 3.2: Boxplot of selected features

- **sbytes:** The box is relatively short, indicating that the values are tightly clustered. The presence of a point outside the box might suggest a residual outlier, although the treatment has reduced the number of outliers.

- **dbytes:** The box is wider, indicating greater variability in the data. The whiskers extend significantly, showing the presence of values well above the third quartile, which could be the result of a more scattered data distribution.

3.3 Feature Engineering

In this section, we focus on data encoding, a crucial step for processing categorical features. In the previous part, we already performed essential analyses such as correlation analysis, feature selection, and the application of principal component analysis (PCA) to better understand our data. To encode categorical variables, we will use techniques such as one-hot encoding or label encoding, which allow us to convert these features into numerical formats suitable for machine learning models.

3.3.1 Normalization and Standardization

Normalization and standardization are two important preprocessing procedures that assist in placing the data on a comparable scale, which can enhance model performance.

3.3.1.1 Normalization

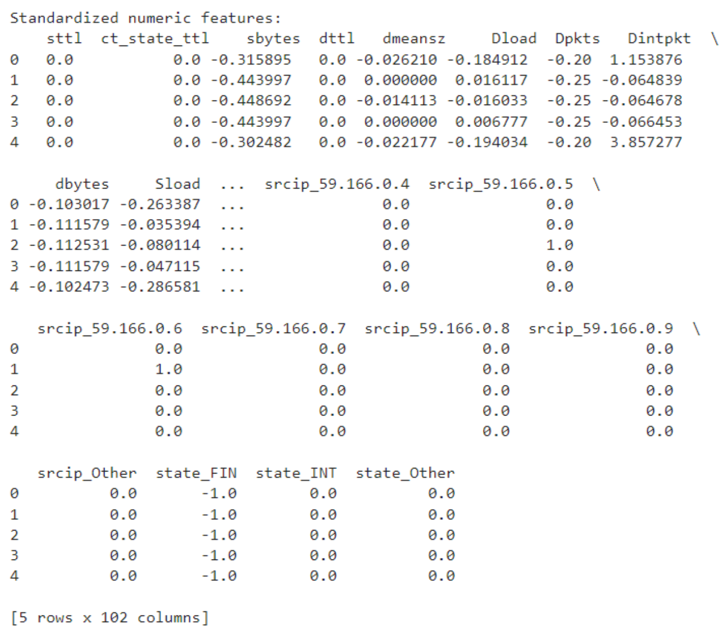
To standardize the data in this step—especially for integer-based numerical features—we employ Min-Max Scaling. By using this method, the values are changed to fall between [0, 1]. We can consider the features as more categorical variables by normalizing them, which can be especially useful in some machine learning scenarios.

3.3.1.2 Standardization

Conversely, standardization is an additional preprocessing method that changes the data to have a mean of 0 and a standard deviation of 1. Many machine learning algorithms depend on the equal contribution of each feature to the model's learning, which is ensured by applying standardization.

3.3.2 Handling Class Imbalance and Visualizing Results

In this section, we focused on addressing class imbalance in our dataset to improve the performance of machine learning models. We employed the Random Oversampling technique from the imblearn library to create a balanced dataset by duplicating instances of the minority



```
Standardized numeric features:
    sttl  ct_state_ttl    sbytes  dttl   dmeansz     Dload  Dpkts   Dintpkt  \
0   0.0           0.0 -0.315895   0.0 -0.026210 -0.184912  -0.20  1.153876
1   0.0           0.0 -0.443997   0.0  0.000000  0.016117  -0.25 -0.064839
2   0.0           0.0 -0.448692   0.0 -0.014113 -0.016033  -0.25 -0.064678
3   0.0           0.0 -0.443997   0.0  0.000000  0.006777  -0.25 -0.066453
4   0.0           0.0 -0.302482   0.0 -0.022177 -0.194034  -0.20  3.857277

     dbytes     Sload  ...  srcip_59.166.0.4  srcip_59.166.0.5  \
0 -0.103017 -0.263387  ...               0.0               0.0
1 -0.111579 -0.035394  ...               0.0               0.0
2 -0.112531 -0.080114  ...               0.0               1.0
3 -0.111579 -0.047115  ...               0.0               0.0
4 -0.102473 -0.286581  ...               0.0               0.0

   srcip_59.166.0.6  srcip_59.166.0.7  srcip_59.166.0.8  srcip_59.166.0.9  \
0               0.0               0.0               0.0               0.0
1               1.0               0.0               0.0               0.0
2               0.0               0.0               0.0               0.0
3               0.0               0.0               0.0               0.0
4               0.0               0.0               0.0               0.0

   srcip_Other  state_FIN  state_INT  state_Other
0          0.0       -1.0        0.0          0.0
1          0.0       -1.0        0.0          0.0
2          0.0       -1.0        0.0          0.0
3          0.0       -1.0        0.0          0.0
4          0.0       -1.0        0.0          0.0

[5 rows x 102 columns]
```

Figure 3.3: Standardized numeric features

class. This approach helps prevent models from being biased toward the majority class and enhances their ability to generalize.

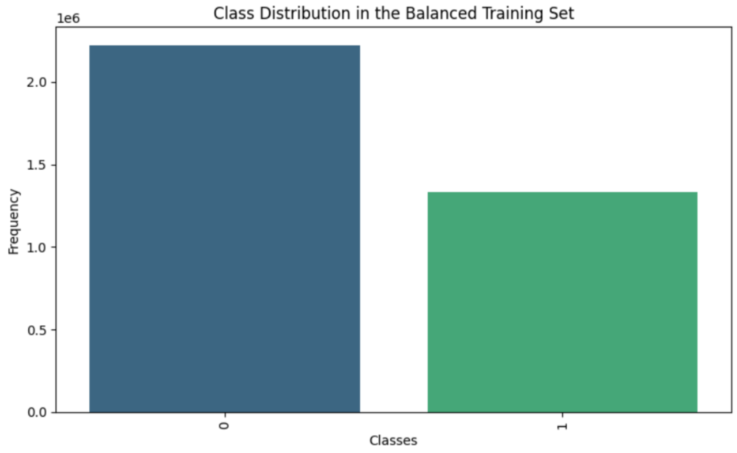
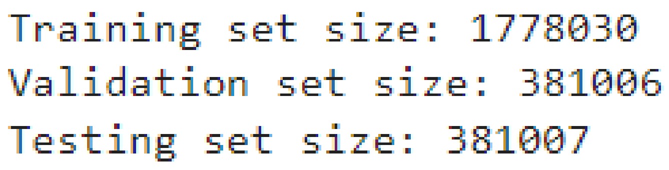


Figure 3.4: Class distribution in the balanced training set

3.3.3 Splitting the data for modeling

To prepare the dataset for modeling, we implemented a systematic approach to split the data into three distinct subsets: training, validation, and testing. Using the train-test-split function from sklearn.model-selection, we allocated 70 percent of the data for training, ensuring that the model has ample data to learn from. The remaining 30 percent was further divided equally into validation and testing sets, each comprising 15 percent of the original dataset. This stratified splitting process helps maintain the proportion of target labels across all subsets, thus preserving the dataset's inherent class distribution.



```
Training set size: 1778030
Validation set size: 381006
Testing set size: 381007
```

Figure 3.5: Splitting data

3.3.4 Conclusion

In this chapter, we meticulously prepared the UNSW-NB15 dataset for modeling by implementing crucial steps such as data cleaning, feature engineering, normalization, and data splitting. We addressed missing values through appropriate imputation techniques, ensuring data integrity and completeness. Outlier detection and management were conducted to enhance the robustness of our analyses. Additionally, we utilized encoding techniques for categorical features to facilitate their integration into machine learning models. Finally, by splitting the dataset into training, validation, and testing subsets, we laid the groundwork for effective model training and evaluation. The methodologies applied in this chapter will not only improve the model's performance but also provide a solid foundation for subsequent analyses.

Chapter 4

Week 4: Data Modeling

Plan

4.1 Introduction

In this chapter, we explore various machine learning (ML) models focusing on both supervised and unsupervised learning techniques, with an emphasis on anomaly detection. The objective is to implement and evaluate different algorithms, assess their performance using relevant metrics, and begin the exploration of deep learning methodologies.

4.2 Supervised Learning with ML

We implemented the following supervised machine learning models:

- **Decision Tree**
- **Random Forest**
- **K-Nearest Neighbors (KNN)**
- **Support Vector Machine (SVM)**
- **Gradient Boosting (XGBoost)**

4.2.1 Decision Tree Classifier

The Decision Tree Classifier is a non-linear model that recursively splits the dataset into subsets based on feature values. It creates branches at each node by selecting the most informative features. The model aims to partition the data in a way that maximizes the homogeneity of the target variable within each subset. To optimize the model's performance, we tuned the following hyperparameters using GridSearchCV:

- **max_depth**: The maximum depth of the tree. This parameter limits the number of splits, helping to avoid overfitting by controlling the complexity of the tree.
- **min_samples_split**: The minimum number of samples required to split an internal node. A higher value prevents the model from creating overly specific branches, reducing the risk of overfitting.
- **min_samples_leaf**: The minimum number of samples required to be at a leaf node. This ensures that each leaf contains enough data to make reliable predictions.

- **criterion**: The function used to measure the quality of a split. Common options are `gini` (Gini impurity) and `entropy` (information gain). These help in determining which feature and value will produce the most informative splits.

We performed a hyperparameter search over the following grid:

```
param_grid_dt = {
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'criterion': ['gini', 'entropy']
}
```

4.2.1.1 Results

After performing GridSearchCV, we selected the best model based on the highest accuracy achieved on the validation set. The Decision Tree Classifier achieved the following hyperparameters:

- ‘max$_{d}epth$‘ : 10‘$min_{s}amples_{s}plit$‘ : 2

- ‘min$_{s}amples_{l}eaf$‘ : 1‘$criterion$‘ :$'\ gini'$

This model resulted in an **accuracy of 75.65%**, **precision of 76.43%**, **recall of 75.65%**, and an **ROC-AUC score of 0.88**. These results suggest that while the Decision Tree performed reasonably well, its ability to generalize to unseen data was somewhat limited compared to more complex models like Random Forest and XGBoost.

4.2.2 Random Forest Classifier

The Random Forest Classifier is an ensemble method that combines multiple Decision Trees to improve accuracy and reduce overfitting. The model was tuned with the following hyperparameters:

- `n_estimators`: The number of trees in the forest.
- `max_depth`: Maximum depth of each tree.
- `min_samples_split`: Minimum number of samples required to split an internal node.
- `min_samples_leaf`: Minimum number of samples required to be at a leaf node.

- `bootstrap`: Whether bootstrap sampling is used when building trees.

We used the following grid for hyperparameter tuning:

```
param_grid_rf = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1],
    'bootstrap': [True]
}
```

4.2.2.1 Results

After GridSearchCV, the best model achieved the following hyperparameters:

- `n_estimators`: 200
- `max_depth`: 10
- `min_samples_split`: 2
- `min_samples_leaf`: 1
- `bootstrap`: True

The Random Forest Classifier achieved these results:

- **Accuracy:** 80.48%
- **Precision:** 81.26%
- **Recall:** 80.48%
- **ROC-AUC Score:** 0.9627

4.2.3 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a non-parametric model that classifies data points based on the majority class of their nearest neighbors. We tuned the following hyperparameters:

- `n_neighbors`: The number of neighbors to consider for classification.

- `weights`: The weight function used in prediction, either `uniform` or `distance`.
- `metric`: The distance metric used for measuring the distance between data points, typically `euclidean`.

We used the following grid for tuning the model:

```
param_grid_knn = {
    'n_neighbors': [3, 5],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean']
}
```

4.2.3.1 Results

The best model selected from GridSearchCV achieved the following hyperparameters:

- `n_neighbors`: 5
- `weights`: `uniform`
- `metric`: `euclidean`

The KNN model achieved these results:

- **Precision:** 74.35%
- **Recall:** 73.11%
- **ROC-AUC Score:** 0.9113

4.2.4 Support Vector Machine (SVM)

The Support Vector Machine (SVM) is a powerful classifier that finds the optimal hyperplane separating the data. We tuned the following hyperparameters:

- **‘C‘**: Regularization parameter that controls the trade-off between achieving a low error on the training data and maintaining a smooth decision boundary.
- **‘kernel‘**: The kernel used to transform the data, typically ‘linear‘, ‘rbf‘, or ‘poly‘.
- **‘gamma‘**: Kernel coefficient for ‘rbf‘, ‘poly‘, and ‘sigmoid‘ kernels.

We used the following grid for tuning the SVM:

```
param_grid_svm = {
    'C': [1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}
```

4.2.4.1 Results

The best model for the SVM achieved the following hyperparameters:

- ‘C‘: 1
- ‘kernel‘: ’rbf’
- ‘gamma‘: ’scale’

The SVM model achieved these results:

- **Accuracy:** 78.42%
- **Precision:** 79.23%
- **Recall:** 78.42%
- **ROC-AUC Score:** 0.9235

4.2.5 Gradient Boosting (XGBoost)

Gradient Boosting, specifically **XGBoost**, is an ensemble technique that builds models sequentially, with each new model correcting the errors of the previous ones. We tuned the following hyperparameters:

- `n_estimators`: The number of boosting rounds (trees).
- `max_depth`: Maximum depth of trees in the boosting model.
- `learning_rate`: Step size shrinking to prevent overfitting.
- `subsample`: Proportion of training data used for each boosting round.
- `colsample_bytree`: Fraction of features used for each tree.

We used the following grid for tuning the XGBoost model:

```
param_grid_xgb = {
    'n_estimators': [100, 200],
    'max_depth': [6, 10],
    'learning_rate': [0.1, 0.05],
    'subsample': [0.8],
    'colsample_bytree': [0.8]
}
```

4.2.5.1 Results

The best model for XGBoost achieved the following hyperparameters:

- `n_estimators`: 200
- `max_depth`: 6
- `learning_rate`: 0.1
- `subsample`: 0.8
- `colsample_bytree`: 0.8

XGBoost achieved the following performance metrics:

- **Accuracy:** 81.67%
- **Precision:** 83.67%
- **Recall:** 81.67%
- **ROC-AUC Score:** 0.9772

4.2.6 Summary of Model Performance

The performance of each model is summarized in the table below:

Model	Accuracy	Precision	Recall	F1 Score	ROC-AUC
Decision Tree	75.65%	76.43%	75.65%	75.59%	0.88
Random Forest	80.48%	81.26%	80.48%	80.58%	0.9627
KNN	73.11%	74.35%	73.11%	73.39%	0.9113
SVM	78.42%	79.23%	78.42%	78.83%	0.9235
XGBoost	81.67%	83.67%	81.67%	81.41%	0.9772

Table 4.1: Summary of Model Performance

4.2.7 Conclusion

In conclusion, **XGBoost** emerged as the top performer overall, achieving the highest values for **accuracy**, **precision**, **recall**, and **ROC-AUC**. It was closely followed by **Random Forest**, which also demonstrated strong performance across these metrics. While both **Decision Tree** and **KNN** yielded reasonable results, they were less effective compared to the more sophisticated ensemble models. The performance of each model was significantly influenced by its hyperparameters, with **XGBoost** benefiting from the most optimal tuning for this dataset.

4.3 Unsupervised Learning with Machine Learning

We implemented the following unsupervised learning models for clustering and anomaly detection on the dataset:

- **K-Means Clustering**
- **Agglomerative Clustering**

4.3.1 K-Means Clustering

- **Overview:**

  K-Means is a partition-based clustering algorithm that divides the dataset into a predefined number of clusters. In this implementation, the number of clusters was set to 3. The algorithm works by iteratively assigning data points to the nearest cluster centroid and updating the centroids until convergence is achieved.

- **Hyperparameters:**

  - **n_clusters**: 3 (for 3 species).

  - **random_state**: 42 (for reproducibility).
  - **n_init**: 10 (multiple initializations).
  - **max_iter**: 300 (default for convergence).
  - **init**: k-means++ (optimal centroid initialization).

- **Results:**
  - **ARI:** 0.73 (moderate alignment with true labels).
  - **Silhouette Score:** 0.55 (moderate separation).
  - **Cluster Purity:** 0.89 (89% homogeneity within clusters).

4.3.2 Agglomerative Clustering

- **Overview:**

  Agglomerative Clustering is a hierarchical clustering algorithm that builds a tree-like structure, called a dendrogram, by recursively merging the closest pairs of clusters. Unlike K-Means, this algorithm does not require the number of clusters to be predefined; however, in this implementation, the number of clusters was set to 3. The algorithm starts with each data point as its own cluster and iteratively merges the closest clusters based on a specified distance metric until the desired number of clusters is reached.

- **Hyperparameters:**
  - **n_clusters**: Set to 3 to match the true number of species.
  - **affinity**: Euclidean distance used to compute the linkage between clusters.
  - **linkage**: Ward linkage criterion, which minimizes the variance of merged clusters.
  - **compute_full_tree**: Set to False for faster computation by stopping the tree-building process early.

Results:

- **ARI:** 0.73 (similar to KMeans).
- **Silhouette Score:** 0.55 (same as KMeans).
- **Cluster Purity:** 0.8933 (slightly better than KMeans).

4.3.3 Evaluation Metrics

- **Adjusted Rand Index (ARI):**
  - KMeans and Agglomerative: **0.73**, indicating good alignment with true labels.
- **Silhouette Score:**
  - Both methods: **0.55**, indicating moderate separation between clusters.
- **Cluster Purity:**
  - KMeans: **0.89**, Agglomerative: **0.8933**, indicating slightly better grouping with Agglomerative Clustering.

4.3.4 Visualization and Insights

- **Dendrogram Visualization:**
  - The dendrogram generated by Agglomerative Clustering illustrates the hierarchical structure of clusters. It shows how clusters are progressively merged, and cutting the tree at a specific level yields the desired number of clusters.

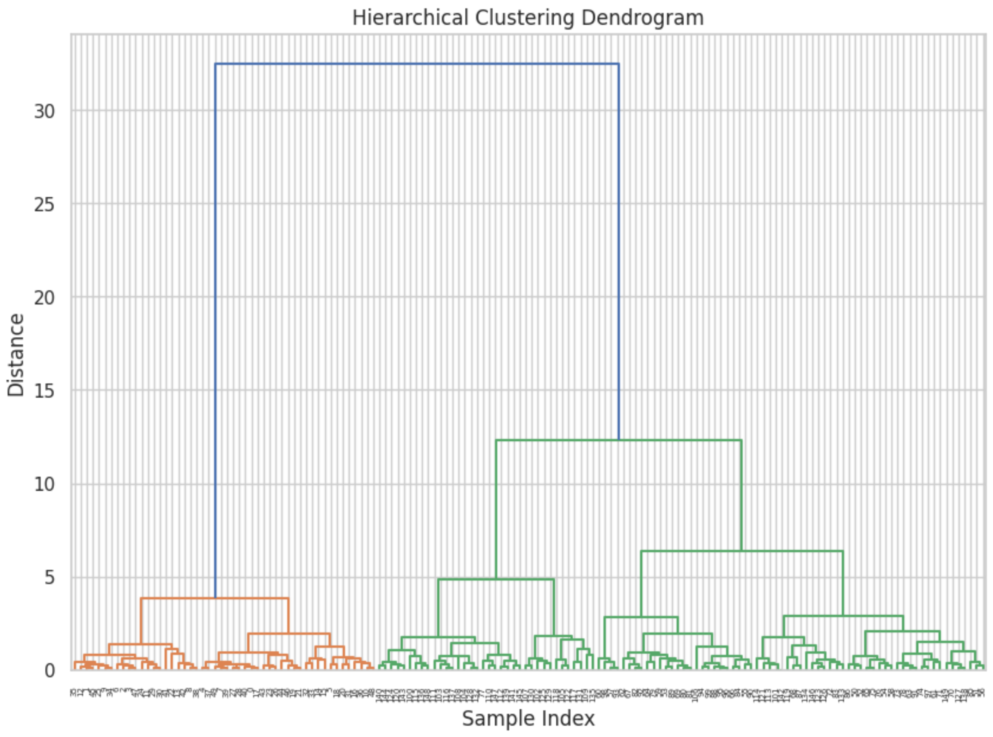


Figure 4.1: Dendrogram for Agglomerative Clustering showing the hierarchical cluster formation.

- **PCA Visualization:**
  - Scatter plots for both K-Means and Agglomerative Clustering show the clustering results in a 2D space. The plots reveal moderate separation between clusters, with some overlap.

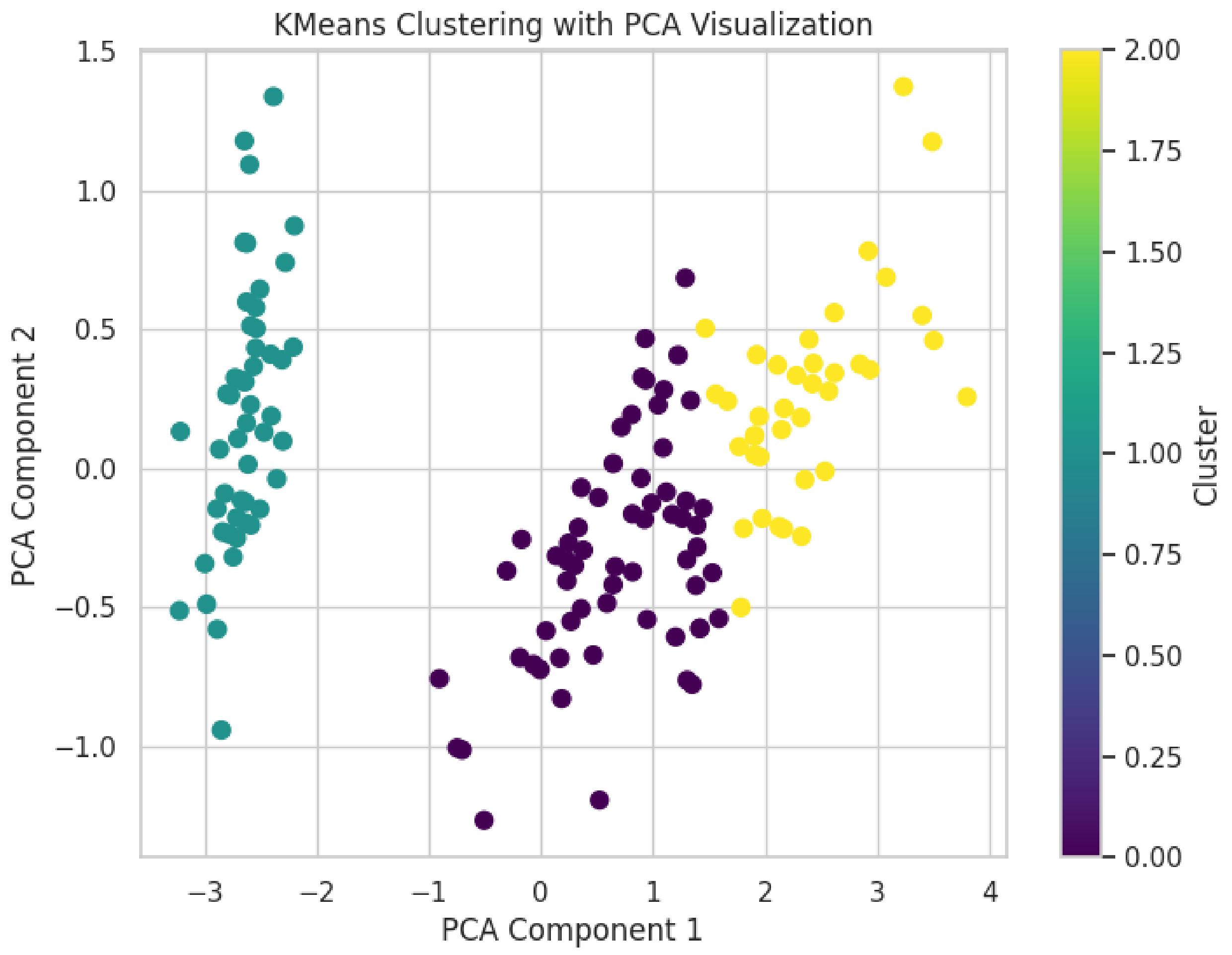


Figure 4.2: PCA Visualization of K-Means and Agglomerative Clustering showing moderate cluster separation.

4.3.5 Comparison of K-Means and Agglomerative Clustering

The following table summarizes the key evaluation metrics for both K-Means and Agglomerative Clustering, highlighting the performance of each model on the Iris dataset.

Metric	**K-Means**	**Agglomerative Clustering**
Adjusted Rand Index (ARI)	0.73	0.73
Cluster Purity	0.89	0.8933
Silhouette Score	0.55	0.55

Table 4.2: Comparison of K-Means and Agglomerative Clustering Results

4.3.6 Conclusion

Both K-Means and Agglomerative Clustering performed similarly on our dataset, with identical Adjusted Rand Index (ARI) scores of 0.73, indicating a good alignment with the true species labels. In terms of cluster purity, Agglomerative Clustering achieved a slightly higher score of 0.8933, compared to 0.89 for K-Means, suggesting that Agglomerative Clustering was marginally better at grouping data points from the same species together. Both models also yielded the same Silhouette Score of 0.55, which reflects moderate cluster separation with some overlap between the clusters. In terms of efficiency, K-Means is faster and more scalable, making it suitable for larger datasets or situations where computational efficiency is critical. On the other hand, Agglomerative Clustering offers a hierarchical perspective, which is valuable for understanding the relationships between clusters and can be particularly useful when analyzing nested structures within the data.

4.4 Initial Evaluation of ML Models

In the evaluation of the machine learning models implemented in this project, we employed a comprehensive range of metrics to assess their performance. The following key metrics were utilized:

- **Accuracy:** This metric provides a general measure of how often the model correctly predicts the target variable across all instances.
- **Precision:** This metric indicates the proportion of true positive predictions among all positive predictions made by the model.
- **Recall (Sensitivity):** This metric measures the proportion of true positive predictions among all actual positive instances. It is particularly important in applications where missing a positive case is more detrimental than false alarms.
- **F1-Score:** The F1-score is the harmonic mean of precision and recall, providing a balance between the two. This metric is especially useful when seeking a single measure of performance that accounts for both false positives and false negatives.
- **ROC-AUC Curve:** The Receiver Operating Characteristic (ROC) curve and the Area Under the Curve (AUC) metric assess the trade-off between sensitivity and specificity across various thresholds. This is essential for understanding model performance in imbalanced datasets, allowing for more nuanced decision-making.
- **Loss Plots:** We also monitored loss plots during training to evaluate the models' learning behavior. These plots help identify overfitting or underfitting, guiding adjustments to model architecture or training parameters.

By employing this diverse set of metrics, we aimed to provide a thorough and reflective evaluation of the models' capabilities, ensuring they are well-suited for real-world applications. Each metric offers unique insights, collectively enhancing our understanding of model performance and areas for improvement.

4.5 Supervised Learning Models

In this section, we evaluated four supervised machine learning models: Decision Trees, Random Forests, K-Nearest Neighbors (KNN), and XGBoost. The evaluation metrics used include accuracy, precision, recall, F1-score, and ROC-AUC.

Model	Accuracy	Precision	Recall	F1 Score	ROC-AUC
Decision Tree	75.65%	76.43%	75.65%	75.59%	0.88
Random Forest	80.48%	81.26%	80.48%	80.58%	0.9627
KNN	73.11%	74.35%	73.11%	73.39%	0.9113
SVM	78.42%	79.23%	78.42%	78.83%	0.9235
XGBoost	81.67%	83.67%	81.67%	81.41%	0.9772

Table 4.3: Summary of Model Performance

4.5.1 Observations

- **Decision Tree and Random Forest:** Both models showed similar performance with an accuracy of 68.09%. High recall rates (nearly 100%) indicate they effectively identified positive instances, but their ROC-AUC scores suggest limited ability to discriminate between classes.

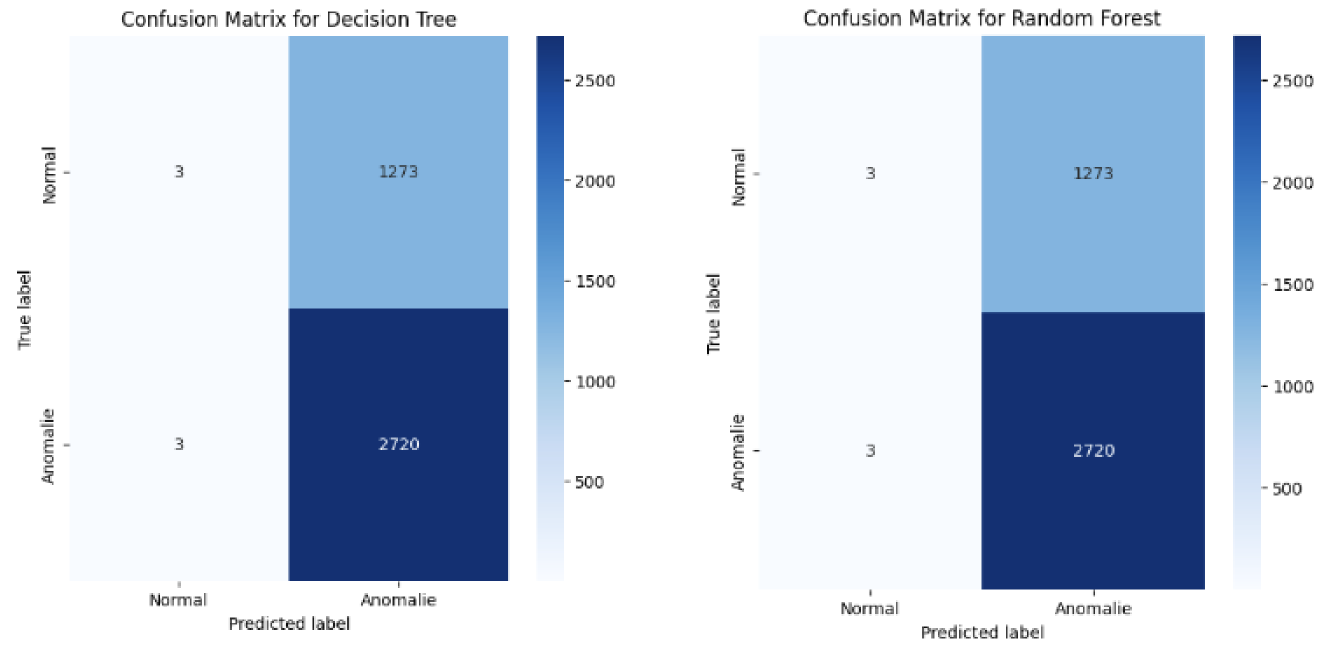


Figure 4.3: Confusion Matrix for Decision Tree and Random Forest

- **KNN:** This model exhibited the lowest performance with an accuracy of 62.57%. The precision and recall values reflect its struggle to balance false positives and false negatives.

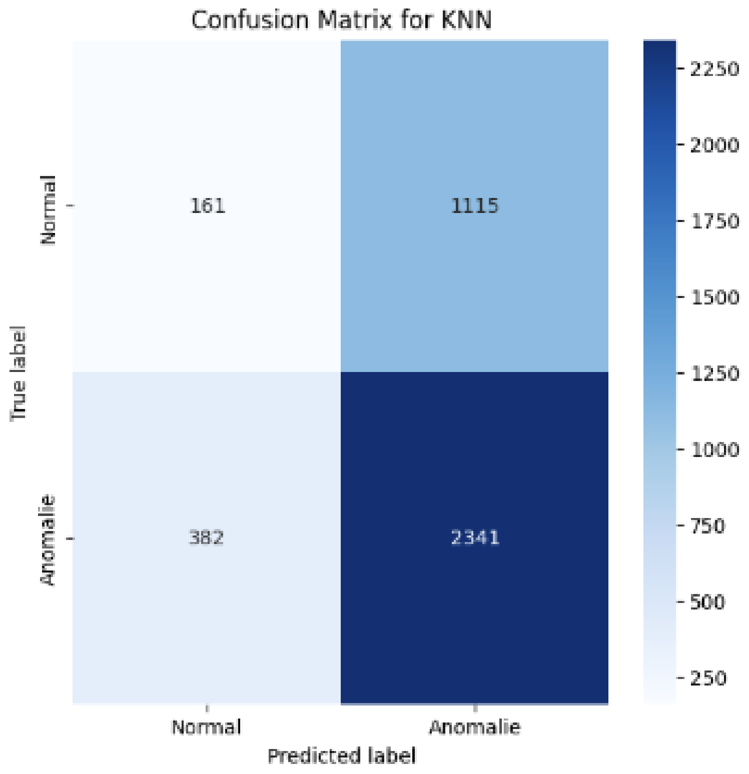


Figure 4.4: Confusion Matrix for KNN

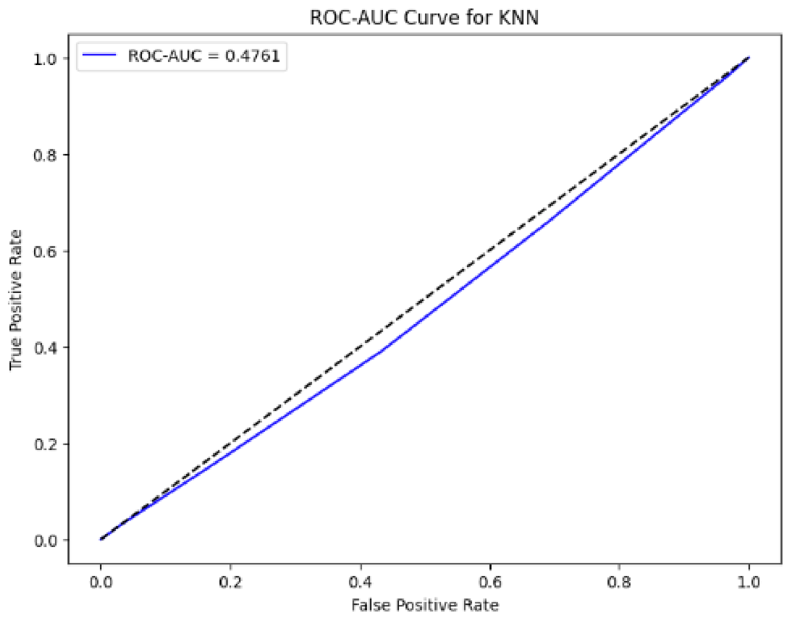


Figure 4.5: ROC-AUC Curve for KNN

- **XGBoost:** This model showed comparable performance to the other ensemble methods, achieving perfect recall but slightly lower precision. This indicates a potential issue with overfitting, as it might be classifying almost everything as positive.

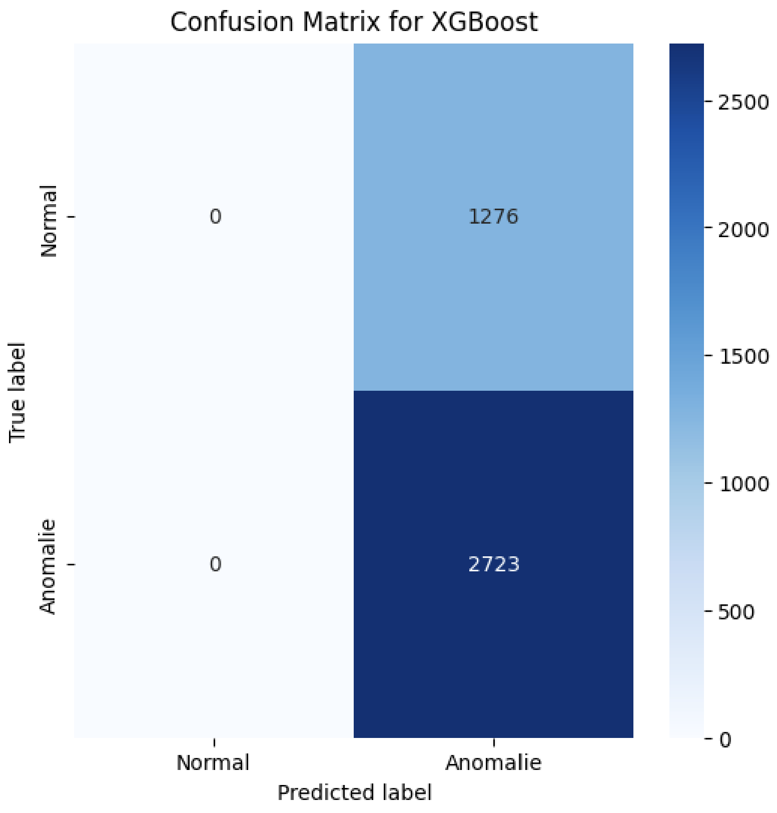


Figure 4.6: Confusion Matrix for XGBoost

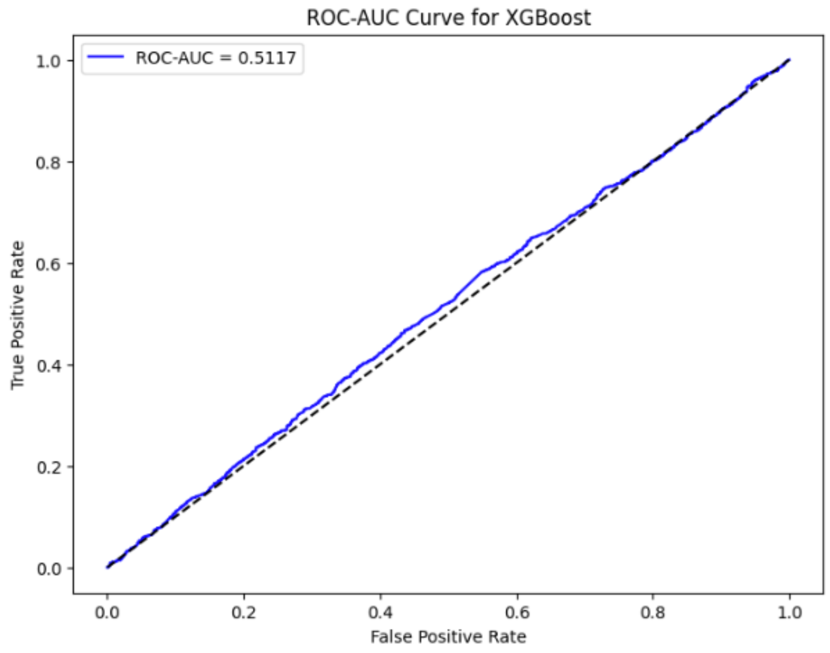


Figure 4.7: ROC-AUC Curve for XGBoost

4.5.2 Comparison to Previous Works:

These results align with existing literature that emphasizes the effectiveness of ensemble methods, especially in scenarios with imbalanced datasets.

Notably, the high recall rates indicate strong performance in identifying positive cases, similar to findings in prior studies. However, improvements are necessary to enhance precision and overall robustness.

4.5.3 Conclusion:

The evaluation indicates that while the models effectively identify positive instances, there is room for improvement in balancing precision and recall. Further tuning and feature engineering are needed to enhance overall model performance, especially for KNN, which underperformed relative to the other methods.

4.6 Begin Deep Learning Modeling

4.6.1 Environment Setup

For the deep learning component, we established a suitable environment utilizing Keras and TensorFlow. This included:

- Installation of necessary libraries (Keras, TensorFlow, NumPy, etc.).
- Verification of GPU support to accelerate model training.

4.6.2 Implementation of a Basic MLP Model

We began by transforming categorical features using OneHotEncoder and then implemented a deep learning model with convolutional layers, LSTM blocks, and dense layers. The architecture utilized the Keras Sequential API and TensorFlow for backend computation.

4.6.2.1 Data Preprocessing

The data was preprocessed using the `ColumnTransformer` to apply OneHotEncoding to categorical variables, and the following transformations were applied:

4.6.2.2 Model Architecture

The model architecture consists of several layers, including convolutional blocks, LSTM layers, and dense layers. We also used batch normalization and dropout to enhance generalization. Below is the description of the layers used in the model:

- **First LSTM Block**: An LSTM layer with 16 units, set to return sequences for further processing.
- **Second Convolutional Block**: A Conv1D layer with 32 filters and a kernel size of 3, followed by MaxPooling1D with pool size 2. BatchNormalization is applied after the MaxPooling layer.
- **Second LSTM Block**: Another LSTM layer, this time with 32 units, also returning sequences.
- **Third Convolutional Block**: A Conv1D layer with 64 filters and a kernel size of 5, followed by MaxPooling1D with pool size 2. BatchNormalization is applied again to normalize the output.

- **Third LSTM Block**: The final LSTM layer with 64 units, which processes the features sequentially, but this time without returning sequences.
- **Dense Layer**: A fully connected Dense layer with 64 units and ReLU activation.
- **Dropout Layer**: A Dropout layer with a dropout rate of 0.2 to prevent overfitting.
- **Output Layer**: A final Dense layer with 10 units and a softmax activation, which outputs a probability distribution across 10 possible classes for multi-class classification.

4.6.2.3 Model Training and Evaluation

The model was trained for 5 epochs with a batch size of 256. The training included both training and validation sets to monitor overfitting. The results of training were as follows:

- **Test Loss**: 0.0556
- **Test Accuracy**: 97.77%
- **Test Precision**: 99.00%
- **Test Recall**: 96.90%

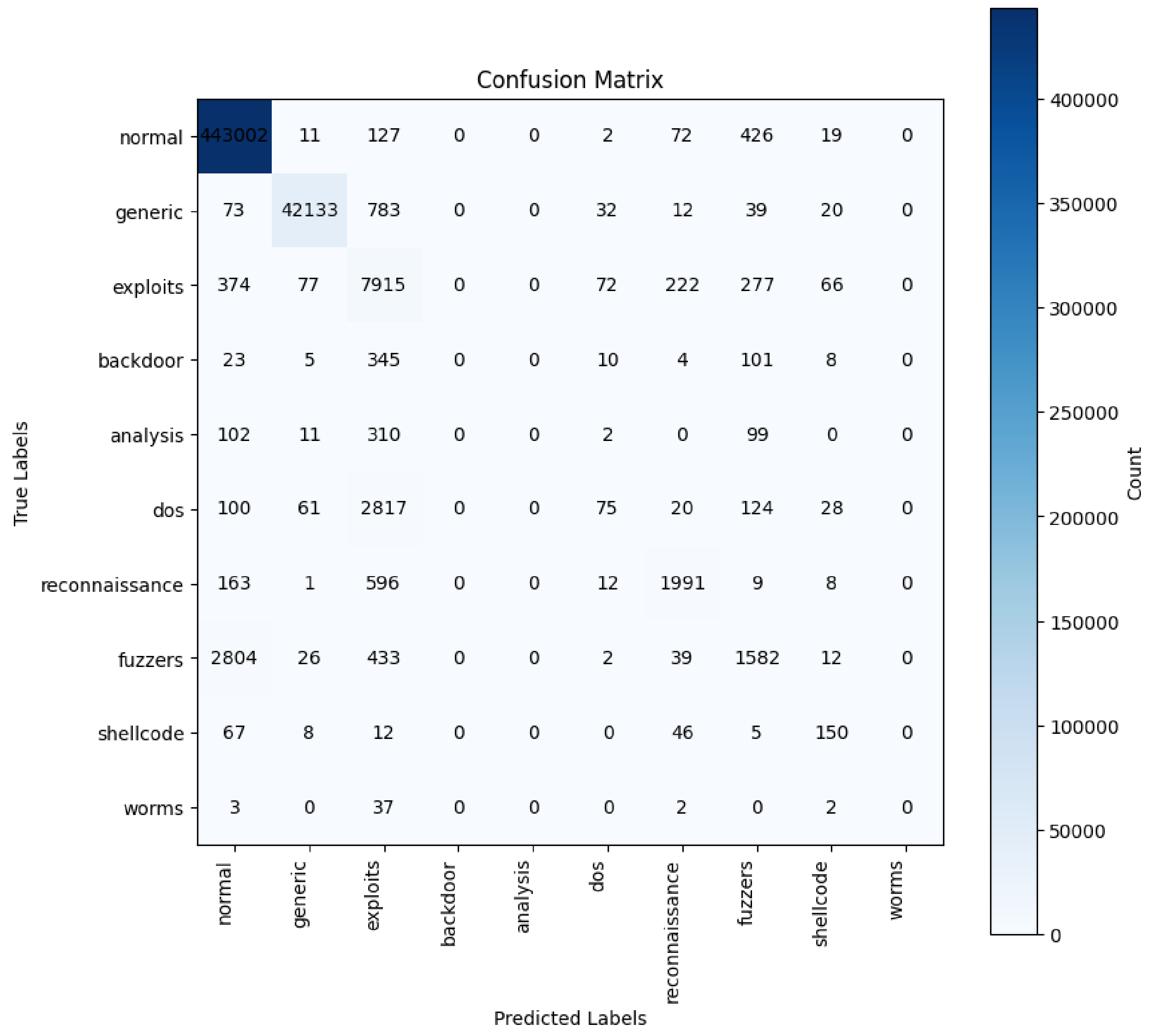
4.6.2.4 Cross-Validation and K-Fold Evaluation

We utilized Stratified K-Fold cross-validation to ensure the model's robustness. The average K-fold cross-validation results on the validation set are:

- **Loss**: 0.0565
- **Accuracy**: 97.74%
- **Precision**: 98.99%
- **Recall**: 96.87%

4.6.3 Confusion Matrix

The confusion matrix was generated to evaluate the model's performance in terms of true positives, true negatives, false positives, and false negatives. The confusion matrix shows the distribution of predicted and actual values across the 10 classes.



True \ Predicted	normal	generic	exploits	backdoor	analysis	dos	reconnaissance	fuzzers	shellcode	worms
normal	443002	11	127	0	0	2	72	426	19	0
generic	73	42133	783	0	0	32	12	39	20	0
exploits	374	77	7915	0	0	72	222	277	66	0
backdoor	23	5	345	0	0	10	4	101	8	0
analysis	102	11	310	0	0	2	0	99	0	0
dos	100	61	2817	0	0	75	20	124	28	0
reconnaissance	163	1	596	0	0	12	1991	9	8	0
fuzzers	2804	26	433	0	0	2	39	1582	12	0
shellcode	67	8	12	0	0	0	46	5	150	0
worms	3	0	37	0	0	0	2	0	2	0

Figure 4.8: Confusion Matrix for Model Performance

4.6.4 Conclusion

The model achieved excellent performance with high accuracy, precision, and recall. The K-fold cross-validation results confirm the robustness of the model, and the confusion matrix visualizes the model's ability to distinguish between different classes.

4.7 Federated Learning Implementation

4.7.1 Overview

In this section, we implement a federated learning approach to train a deep learning model across multiple clients. The model training occurs locally on each client using a subset of the training data, and then the weights are aggregated to update the global model. This approach helps maintain data privacy and reduces the need to transfer sensitive data.

4.7.2 Data Partitioning

We begin by dividing the training data into multiple subsets, each corresponding to a different client. The number of clients is defined as *num_clients*, and the training data *x_train* and labels *y_train* are split equally among these clients. Each client will train the model locally with its own data subset.

- **Data Split**: The training data and labels are divided into *num_clients* subsets using `np.array_split`.

4.7.3 Model Architecture

Each client in the federated learning setup trains an identical deep learning model. The model consists of convolutional blocks followed by LSTM layers and dense layers. The architecture is designed to handle sequential data and perform multi-class classification.

- **Conv1D Layers**: These layers are used for extracting features from the sequential data.
- **MaxPooling1D**: MaxPooling is applied after each convolutional layer to reduce the spatial dimensions.
- **LSTM Layers**: These layers help the model learn temporal dependencies from the sequential data.
- **Dense Layer**: Fully connected layer used for classification.
- **Dropout Layer**: Applied to avoid overfitting by randomly setting some of the units to zero during training.
- **Softmax Activation**: The final layer uses softmax activation to output a probability distribution across 10 classes.

4.7.4 Federated Learning Process

The federated learning process consists of the following steps:

1. **Client Training**: Each client trains its local model using its own subset of data. The local models are initialized with the global model's weights at the start of each round.
2. **Weight Aggregation**: After local training, the weights of each client's model are sent back to the central server, where they are aggregated by computing the average of all the client weights.
3. **Global Model Update**: The global model is updated with the aggregated weights and distributed back to the clients for the next round of training.
4. **Model Evaluation**: The global model's performance is evaluated after each round on the test set, and the accuracy is reported.

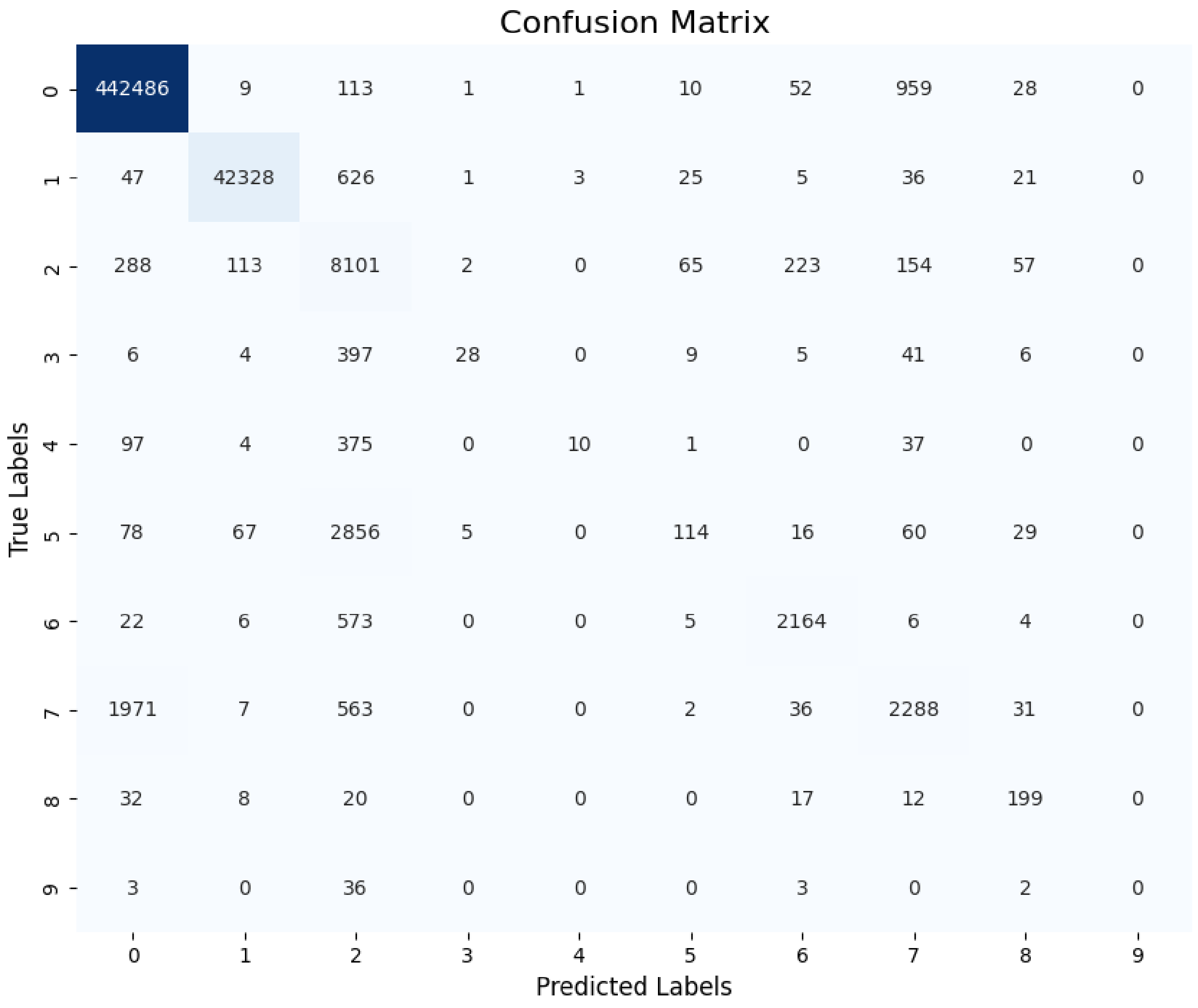
4.7.5 Training and Results

The federated learning process is carried out over 10 rounds with each round consisting of 5 epochs. The following are the results after the completion of the 10 rounds:

- **Test Accuracy**: 97.97%
- **Test Loss**: 0.0506
- **Test Precision**: 99.06%
- **Test Recall**: 97.18%

4.7.6 Confusion Matrix

The confusion matrix was generated to evaluate the performance of the global model after training. Below is the confusion matrix which shows the true positives, false positives, true negatives, and false negatives across the 10 classes.



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	442486	9	113	1	1	10	52	959	28	0
1	47	42328	626	1	3	25	5	36	21	0
2	288	113	8101	2	0	65	223	154	57	0
3	6	4	397	28	0	9	5	41	6	0
4	97	4	375	0	10	1	0	37	0	0
5	78	67	2856	5	0	114	16	60	29	0
6	22	6	573	0	0	5	2164	6	4	0
7	1971	7	563	0	0	2	36	2288	31	0
8	32	8	20	0	0	0	17	12	199	0
9	3	0	36	0	0	0	3	0	2	0

Figure 4.9: Confusion Matrix for Model Performance

Conclusion

This project has undertaken a thorough investigation into the critical domain of network security, with a particular focus on the challenges posed by network anomalies and intrusions. Through an extensive literature review, we have delineated the fundamental concepts of network anomalies and intrusions, elucidating their causes and providing illustrative examples.

Our exploration of various anomaly detection and intrusion detection systems has underscored the pivotal role of advanced methodologies, including machine learning (ML), deep learning (DL), and federated learning (FL), in fortifying network security. These sophisticated approaches not only enhance the accuracy of threat detection but also facilitate real-time responses to potential security breaches, thereby significantly improving the resilience of network infrastructures.

Furthermore, we have critically evaluated prominent datasets such as UNSW-NB15 and IRIS, which serve as essential resources for training and assessing machine learning models in the context of network intrusion detection. These datasets reflect contemporary network traffic patterns and encompass a diverse array of attack vectors, providing a robust foundation for ongoing research and development in this vital area.

In summary, the findings of this project contribute to the broader understanding of network security challenges and highlight the necessity for continuous innovation in detection methodologies. As cyber threats evolve, so too must our strategies for defense, ensuring that we remain one step ahead in safeguarding our digital environments.

Bibliography

[1] Nour Moustafa and Jill Slay. *The UNSW-NB15 dataset for network-based intrusion detection systems (NIDS) benchmarking*. Proceedings of the 2015 Military Communications and Information Systems Conference (MilCIS), IEEE, 2015.

[2] Ahmad Lashkari, et al. *A comprehensive review of network traffic analysis and performance evaluation metrics for intrusion detection*. Journal of Network and Computer Applications, 110 (2018): 101-111.

[3] Wael L. Al-Yaseen, Zulaiha Ali Othman, and Mohd Nazri Mohd Warip. *Hybrid intelligent intrusion detection system using signature-based and anomaly-based techniques*. Neural Computing and Applications, 30 (2018): 2973-2995.

[4] Manfred Ring, et al. *A survey of network-based intrusion detection data sets*. Computers & Security, 86 (2019): 147-167.

[5] `https://docs.zeek.org/en/master/about.html`

[6] `https://labelstud.io/blog/`

[7] `https://medium.com/@subrata.maji16/building-an-intrusion-detection-system-on-unsw-nb1`

[8] Nour Moustafa and Jill Slay. *The UNSW-NB15 dataset for network-based intrusion detection systems (NIDS) benchmarking*. Proceedings of the 2015 Military Communications and Information Systems Conference (MilCIS), IEEE, 2015.

[9] Manfred Ring, et al. *A survey of network-based intrusion detection data sets*. Computers & Security, 86 (2019): 147-167.

[10] Nick Koroniotis, et al. *Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset*. Future Generation Computer Systems, 100 (2019): 779-796.

[11] Ahmad Lashkari, et al. *A comprehensive review of network traffic analysis and performance evaluation metrics for intrusion detection.* Journal of Network and Computer Applications, 110 (2018): 101-111.

[12] *Zeek Documentation.* Available: `https://docs.zeek.org/en/master/about.html`

[13] *Label Studio Blog.* Available: `https://labelstud.io/blog/`

[14] *Building an Intrusion Detection System on UNSW-NB15 Dataset Based on Machine Learning Algorithm.* Available: `https://medium.com/@subrata.maji16/building-an-intrusion-detection-system-on-unsw-nb15-dataset-based-on-machine-learnin`

[15] The class distribution of the UNSW-NB15 dataset. *ResearchGate.* Available: `https://www.researchgate.net/figure/The-class-distribution-of-the-UNSW-NB15-dataset_tbl2_333573656`.

[16] Zhang, C., et al. *A novel ensemble model for intrusion detection.* Journal of Big Data, 7 (2020): 43. Available: `https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00379-6`.

[17] Subrata Maji. *Building an Intrusion Detection System on UNSW-NB15 Dataset based on Machine Learning Algorithm.* Medium. Available: `https://medium.com/@subrata.maji16/building-an-intrusion-detection-system-on-unsw-nb15-dataset-based-on-machine-learnin`