UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Intrusion Detection Systems using Machine Learning

Diploma Thesis

Ntouvlis Dimitrios

Supervisor: Aspassia Daskalopulu

January 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Συστήματα Ανίχνευσης Εισβολών με χρήση Μηχανικής Μάθησης

Διπλωματική Εργασία

Ντούβλης Δημήτριος

Επιβλέπουσα: Ασπασία Δασκαλοπούλου

Ιανουάριος 2025

Approved by the Examination Committee:

Supervisor **Aspassia Daskalopulu**
Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Dimitrios Katsaros**
Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Alexandros Chronaios**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Acknowledgements

Upon completion of this dissertation I would like to thank my supervisor, Associate Prof. Aspassia Daskalopulu for the opportunity to work with creative freedom which culminated in such an interesting project.

Moreover, I owe special thanks to my friends, family and Christine who have all been very supportive, encouraging, and were always there reminding me to keep my head up throughout this long academic journey.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Ntouvlis Dimitrios

Diploma Thesis

Intrusion Detection Systems using Machine Learning

Ntouvlis Dimitrios

Abstract

In the modern era, cybersecurity threats have become increasingly sophisticated, making the development and evolution of robust Intrusion Detection Systems (IDS) a necessity. This thesis explores the development and evaluation of IDS using advanced machine learning techniques, with a particular focus on detecting Distributed Denial of Service (DDoS) attacks. Utilizing the CIC-DDoS2019 dataset and extensive preprocessing, label mapping, feature scaling, balancing and sampling, the study implements multiple machine learning algorithms, including Random Forest, XGBoost, K-Nearest Neighbors, and Multi-Layer Perceptrons. The research is structured into three experiments: (1) training on mutually represented classes across datasets, (2) merging datasets for a unified representation, and (3) binary classification to differentiate between benign and malicious traffic.

The results demonstrate the different model structures and performances, emphasizing on real world conditions, with XGBoost consistently achieving high performance across precision, recall, and AUC metrics. Analysis and comparison with existing research highlights the importance of excellent preprocessing and feature engineering in achieving the best stats. This study contributes to the field by providing three different comprehensive methodologies and approaches for IDS development and underlines the potential of machine learning to address modern cybersecurity challenges.

Keywords:

Intrusion detection, IDS, Machine Learning, Cybersecurity, CIC-DDoS2019, DDoS Detection

Διπλωματική Εργασία

Συστήματα Ανίχνευσης Εισβολών με χρήση Μηχανικής Μάθησης

Ντούβλης Δημήτριος

Περίληψη

Στη σύγχρονη εποχή, οι απειλές για την ασφάλεια στον κυβερνοχώρο έχουν γίνει ολοένα και πιο περίπλοκες, καθιστώντας αναγκαία την εξέλιξη της ανάπτυξης ισχυρών συστημάτων ανίχνευσης εισβολής (IDS). Αυτή η εργασία διερευνά την ανάπτυξη και αξιολόγηση των IDS χρησιμοποιώντας προηγμένες τεχνικές μηχανικής μάθησης, με ιδιαίτερη έμφαση στον εντοπισμό επιθέσεων κατανεμημένης άρνησης υπηρεσίας (DDoS). Χρησιμοποιώντας τα δεδομένα του CIC-DDoS2019 και μετά από εκτεταμένη προεπεξεργασία, χαρτογράφηση ετικετών, εξαγωγή χαρακτηριστικών, εξισορρόπηση και δειγματοληψία, η μελέτη εφαρμόζει πολλαπλούς αλγόριθμους μηχανικής μάθησης, συμπεριλαμβανομένων των Random Forest, XGBoost, K-Nearest Neighbors και Multi-Layer Perceptrons. Η έρευνα ταξινομείται σε τρία πειράματα: (1) εκπαίδευση σε αμοιβαία αντιπροσωπευόμενες τάξεις σε σύνολα δεδομένων, (2) συγχώνευση συνόλων δεδομένων για ενοποιημένη αναπαράσταση και (3) δυαδική ταξινόμηση για τη διαφοροποίηση μεταξύ κανονικής και κακόβουλης ροής.

Τα αποτελέσματα δείχνουν τις διαφορετικές δομές και επιδόσεις του μοντέλου, με το XGBoost να επιτυγχάνει σταθερά υψηλή απόδοση σε μετρήσεις ακρίβειας, ανάκλησης και AUC. Η ανάλυση και η σύγκριση με την υπάρχουσα έρευνα υπογραμμίζει τη σημασία μιας εξαιρετικής προεπεξεργασίας και μηχανικής χαρακτηριστικών για την επίτευξη των καλύτερων στατιστικών. Αυτή η μελέτη συμβάλλει στο πεδίο παρέχοντας τρεις διαφορετικές ολοκληρωμένες μεθοδολογίες και προσεγγίσεις για την ανάπτυξη ενός IDS και υπογραμμίζει τις δυνατότητες της μηχανικής μάθησης για την αντιμετώπιση των σύγχρονων προκλήσεων στον κυβερνοχώρο.

Λέξεις-κλειδιά:

Ανίχνευση εισβολών, IDS, Μηχανική Μάθηση, Κυβερνοασφάλεια, CIC-DDoS2019, Ανίχνευση Κατανεμημένης άρνησης υπηρεσίας

Table of contents

List of figures

List of tables

Abbreviations

IDS	Intrusion Detecion System
IPS	Intrusion Prevention System
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IP	Internet Protocol
ACK	(TCP) Acknowledge
RST	(TCP) Reset
DoS	Denial OF Service
DoH	DNS over HTTPS
TPR	True Positive Rate
FPR	False Positive Rate
TNR	True Negative Rate
FNR	False Negative Rate
AUC	Area Under Curve
ROC	Receiver-Operating characteristic Curve
OSI	Open Systems Interconnection
MTU	Maximum Transmission Unit
HTTPS	Hypertext Transfer Protocol Secure
VPN	Virtual Private Network
SSL	Secure Socket Layer
TLS	Transport Layer Security
C&C	Command and Control
ARP	Address Resolution Protocol
MITM	Man In The Middle
PCC	Pearson Correlation Coefficient

HMM	Hidden Markov Model
GMM	Gaussian Mixture Model
PID	Process ID
SVDD	Support Vector Data Description
RF	Random Forest
XGB	EXtreme Gradient Boosting
MLP	Multi-Layer Perceptron
KNN	K-Nearest Neighbor
DNN	Deep Neural Network
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
LDAP	Lightweight Directory Access Protocol
CIC	Canadian Institute of Cybersecurity

Chapter 1

Introduction

1.1 Problem Statement

The primary challenge in developing an effective IDS is handling the complexity and diversity of modern network traffic. More specifically, attacks almost always vary both in terms of type and magnitude in order to avoid being pattern-recognized by the various IDS mechanisms. Furthermore, research and datasets attempting to cope with this problem often contain disproportionate numbers of benign and malicious instances, making the training of models more difficult. Another common challenge is ensuring that the IDS performs consistently across potentially unseen traffic types.

The thesis reviews machine learning techniques for DDoS detection, relevant datasets that have been used in the bibliography and the models that have been developed. It, then, demonstrates the application of data preprocessing proposed in the literature, further experiments with it on a well known open source dataset, and conducts tests with various ML algorithms to compare their performance across different metrics. This thesis aims to develop trained models for an IDS that are not just theoretically sound but also deployable in real-world scenarios.

Many studies focus on over-optimizing models for high accuracy under ideal conditions, often pre-processing data in ways that artificially inflate performance metrics but may not reflect real-world deployment challenges. This thesis takes a practical approach, aiming to develop models that maintain performance under realistic conditions, while also not being afraid to take risks and include overimbalanced data and imperfections.

1.2 Objectives

The primary objectives of this thesis are to conduct a research on the general theoretical principles of the IDS as a mechanism, (one that is applicable in a real-world environment, rather than one that simply performs well in a controlled, academic setting), dive into the basics of implementation metrics and finally to process the CIC-DDoS2019 dataset effectively, create high-quality input for machine learning models and implement machine learning algorithms, including Random Forest, K-Nearest Neighbors, Multi-Layer Perceptrons, and XGBoost. Moreover, to assess three experimental setups utilizing mutually represented classes across datasets, combined datasets for unified analysis and binary classification of benign versus malicious traffic. These experiments were based on the principle that a practical IDS must detect as many types of attacks as possible, even imperfectly, rather than achieving perfect accuracy on only a subset of threats. Lastly, the thesis aims to compare the results with existing research, highlighting areas of improvement and the potential for future development.

1.3 Structure

The remainder of this thesis is organized as follows:

- Chapter 2: Theoretical concepts on Cyber attacks - A basic overview with greater emphasis on DDoS attack methodologies and tactics
- Chapter 3: Intrusion Detection Systems (IDS) – Covers a theoretical background of types, detection methods and architecture of an IDS
- Chapter 4: Introduction to Performance Metrics and literature review of existing datasets.
- Chapter 5: Methodology and Data Preprocessing – Description of the data preparation process, including label mapping, feature selection and data cleansing.
- Chapter 6: Experimental Setups and Results - Including model training with 3 distinct strategies, performance evaluation, comparison with similar research and final conclusions.

Chapter 2

Theoretical Background

2.1 Introduction

In a world of billions of electronic devices, computer networks have become the backbone of modern life, supporting everything from personal communications to global commerce and critical infrastructure. Our reliance on networks continues to grow as does the complexity and scale of these systems. Organizations and individuals now depend on large dynamic networks that span across the globe. This expansion has introduced new vulnerabilities and introduced the risk of cyberattacks. Millions of cyberattacks happen on a daily basis aiming not only at unaware ordinary Internet users but also at corporations, organizations and even governments. As a result, securing our networks has become a top priority, introducing the need for advanced protection mechanisms. Intrusion Detection Systems (IDS) are mechanisms that can monitor network activity and detect potential threats in real time. As traditional security measures struggle to keep pace with the sophistication of modern attacks, machine learning offers a promising approach to improving IDS, enabling more adaptive, intelligent defenses capable of identifying both known and emerging threats. But how do cyberattacks work? First we will have to talk about threats and vulnerabilities of a system.

2.2 Threats and vulnerabilities

Any weakness or flaw in a system, network, or application that an attacker could use to undermine its security is referred to as a vulnerability. Vulnerabilities can arise from various sources, including bugs in designed software, design flaws, configuration errors, or even

human oversight. For instance, outdated software, unpatched security holes, or weak access control mechanisms create potential entry points for malicious actors. Vulnerabilities themselves do not cause harm directly but serve as openings that can be targeted by external threats.

The possibility that an attacker could exploit a system vulnerability constitutes a threat. Threats can be external (such as cybercriminals, malware, or hackers), or internal (such as disgruntled employees or insider misuse). Threats typically aim to breach system confidentiality, integrity, or availability, leading to disruptions, data theft, or financial loss. In the context of networks and information systems, threats constantly evolve, leveraging new techniques to exploit system weaknesses.

The relationship between vulnerability and threat is intimate. A vulnerability becomes critical when coupled with an active threat that can exploit it. Cybersecurity strategies, such as Intrusion Detection Systems (IDS), are composed to alert and respond to these threats by identifying potential exploits before they cause significant damage. With the integration of machine learning, IDS systems can better analyze the patterns of threats and detect vulnerabilities in real-time, ensuring a more hardy defense mechanism towards both known and emerging risks.

2.3 Network intrusion types

Network intrusion accounts for a major exploitation of a vulnerability and constitutes a key security risk that could result not only in temporary denial but also total refusal of the network service, as well as potential damages in infrastructure. Furthermore, there is the possibility that a local attack occurs when a malevolent user physically enters the system and either attempts to access the actual device or runs a malicious payload. The two main categories of attacks are:

- Passive attacks
- Active attacks

2.3.1 Passive attacks

The attacker is thought to be in a learning or monitoring state when conducting a passive attack. The aim of the attacker is to comprehend the specifics of the potential victim's device,

including its functionality, performance and operation. As a result, the attacker can come up with a more effective and sophisticated attack strategy. Common passive attacks are Mimicry, Port scanning and Idle scanning.

Mimicry

With this technique, a server is operated by a potential attacker which spoofs and ”mimics” the ID of a legitimate server. The dummy server attempts to eavesdrop and sniff communications in the same channel subnet or simply by having a similar domain name with the authenticated server. This could cause clients to initiate communications with the false server, allowing the attacker to acquire user information. The most common example of Mimicry are servers that host websites copying universally known ones in appearance and domain names, such as *Facebook*. Users can easily confuse the same appearance but not legitimate *www.facebok.com* with the original and attempt to login with their credentials. When they enter their credentials, these are copied in plain text and can be used for other, active attack purposes.

Port scanning

Port scanning involves finding open TCP and UDP ports on a host. This will help the attacker monitor, identify and analyze the services that are operating on the victim’s or target’s computer. Port scanning relies on sending client requests to a block of addresses either by swarming all 65536 existing ports, or only some of the most commonly used ones. This procedure not only gives the attacker an advantage in planning the full-scale attack, but also often does not trigger any security mechanisms at all, leaving the victim completely unaware.

The result of port scanning can be the discovery of open, closed, or blocked ports. Open ports indicate that a host has responded that a service is ”listening” on that specific port. For example, open port 22 means that the host is transmitting data via FTP. Closed or not listening ports signify that the host replied negatively, while blocked or dropped ports designate the existence of a Firewall regulating traffic.

Idle Scanning

Attackers can scan a target using this technique without ever sending a packet from their IP address to the target. Instead, the scan can be bounced off a *zombie*, in a side-channel

attack. Reports from intrusion detection systems (IDS) will identify the attacker as a zombie computer. This type of scan not only allows for the discovery of IP-based trust relationships between machines, but it is also incredibly incognito [1].

Idle scans take advantage of the IP header identification field: every IP packet sent from a specific source contains an identifier that uniquely distinguishes the fragments of a source datagram. The assailant initially examines a system using a sequential and foreseeable sequence number (IPID). After identifying an appropriate zombie, the subsequent step involves attempting to create a TCP connection to a specific service (port) on the target system while masquerading as the zombie. This is accomplished by transmitting a SYN packet to the destination computer, impersonating the IP address of the zombie, meaning the source address is configured to match the zombie's IP address. If the port on the target computer is open, it will accept the connection for the service and reply with a SYN/ACK packet to the zombie. The zombie computer will subsequently transmit an RST packet to the target computer (to restore the connection), as it didn't genuinely send the SYN packet initially. As the zombie needed to transmit the RST packet, it will increase its IPID. In this manner, the attacker will be aware whether the target port is accessible. The assailant will transmit an additional packet to the zombie. If the IPID is increased by just one step, the attacker will be aware that the port is not open. The approach presumes that the zombie machine does not engage in any other interactions. If a message is transmitted for different purposes between the attacker's initial interaction with the zombie and the subsequent interaction, apart from the RST message, a false positive will occur.

2.3.2 Active attacks

During an active attack, the assailant has finished gathering intelligence and has completed sufficient reconnaissance on the target device to be prepared to launch their exploit against the victim. Indirect attacks are those in which the attacker compromises and takes control of another machine, turns it into a ”zombie”, and then uses the ”zombie” to proxy all of the attacks through it. Direct attacks, on the other hand, send the exploit from the attacker's machine straight towards the target. From the victim's point of view, the zombie would thus appear to be the attacker's machine. Active attack examples include:

- Botnet: The attacker establishes a *Command and Control* (C&C) server to access all its infected hosts (zombies) to conduct malicious activities.

- Denial of Service (DoS) and Distributed Denial of Service (DDoS): These kinds of attacks mainly focus on draining the resources of a system, therefore preventing authenticated clients from accessing it. DDoS attacks especially make great use of Botnets mentioned above, since they are essentially a DoS attack at a far greater scale.

Malware

Malware refers to any harmful software that can damage any computer system or network. A type of malware can serve various purposes, including deleting data from a hard drive, taking screenshots of the victim's screen, or even setting up a backdoor. Certain categories of malware include simple Worms, Adware/Spyware, Ransomware or Trojan Horse.

Password attacks

In a password assault, the assailant attempts to acquire the password for a user account, an encrypted document, or possibly a network. The objective may change depending on the attacker's goal. In this process, numerous strategies can be employed to try to obtain someone else's password.

- Brute force attack: In such an attack, the protected data is subjected to every possible character combination until the right one is discovered. The amount of time it could take to find the password is a serious drawback, but a brute force attack has the best chance of breaking it.

- Dictionary attack: This method attempts to crack a password by consulting a dictionary of passwords. Since the attack's success depends on the quality of the words in the password file's real word list, this strategy might not always be the best one.

- Keylogger: There are two types of keyloggers: hardware and software. Keyloggers are primarily used to record keystrokes. This can help obtain an unexpected user's password for a protected website, like their e-banking login credentials.

- Trojan Horse: This kind of malware poses as a reliable program or piece of software in order to fool potential victims into installing it. Once setup, the malicious payload remains hidden from the victim by installing itself in the background. A software keylogger set up to remotely send data logs back to the attacker can potentially be used as the payload.

The central idea of these attacks lies in the vulnerability of the human mind to formulate a robust password that includes alphanumeric symbols, both uppercase and lowercase letters, along with numbers and a special character. This can occasionally be an unprofessional method of gathering important information from users, like bank account info, credit card PINs, or other sensitive data. To start, the attacker attempts to appear authentic and offers details that seem genuine from the victim's viewpoint.

Social Engineering attacks

The method of social engineering enables a person to psychologically deceive someone into disclosing sensitive information like user login details or executing a task such like retrieving a file infected with malware onto their device.

- Phishing: This type of attack employs email as the method by which an attacker conceals as a legitimate entity and seeks to acquire essential information like banking passwords.
- Vishing: This method involves using phones, where the attacker attempts to engage in dialogue as an individual representing a credible organization and obtain information from the victim.
- Pharming: This refers to an attack in which a malicious DNS server supplies an incorrect DNS IP for a specific URL that directs the victim to a harmful website.

Spoofing attacks

In a spoofing attack, the attacker poses as a genuine or authorized user or computer by using false information. An attacker must deceive the user into becoming a victim of the attack by trying to deliver a payload or exploit to the victim's system. Common practices include altering the source IP address and source MAC address inside the packets coming from the attacker machine and might occasionally fool a potential victim into believing they are coming from a trustworthy user and conceal the attack's origins. Some spoofing attacks include:

- ARP spoofing
- DNS spoofing

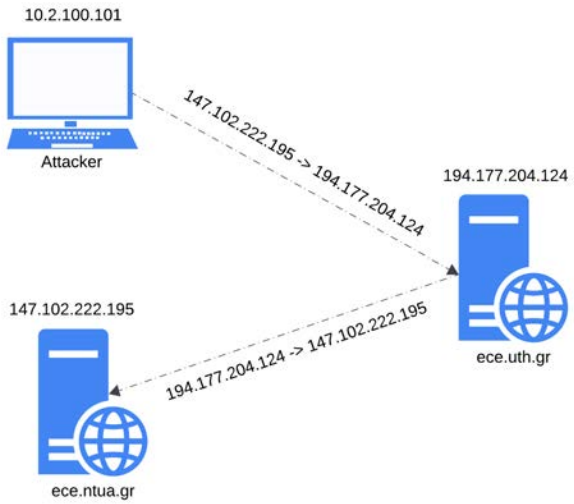

Figure 2.1: Example of an IP spoofing attack

- IP spoofing

Next, we will give an example of a spoofing attack [2], more specifically using the IP spoofing technique. Figure 2.1 shows an attacker sending a custom crafted packet, specifically for this case, to our *e-ce.uth.gr* web server (194.177.204.124) with the aim of crashing *ece.ntua.gr* main server, not exposing him/herself but using *e-ce.uth.gr* as a proxy. The packet is crafted with 147.102.222.195 (*ece.ntua.gr*) IP as source IP header. This IP, though, belongs to the victim's host and not the real IP address of the perpetuator. The moment the server collects the fake packet and before responding, it looks in the sender's IP field and sees the address 147.102.222.195 so it replies to the victim host instead of the attacker. In this case, we would have *ece.uth.gr* server overwhelm *ece.ntua.gr* server with requests, possibly making it unable to respond to legitimate ones.

2.3.3 Denial of Service (DoS) & Distributed Denial of Service (DDoS)

In this study, a DDoS dataset has been utilized for experimenting, so it is crucial that emphasis has been given on understanding the specifics and the techniques used behind these attacks. As already mentioned in 2.3.2, there is no DDoS attack without an infested Botnet behind it, due to the massive scale required to take down large organizations or quite hardened infrastructure. If someone wants to attack a large server, such as a corporation or state owned one, it is not enough to send requests from ten or twenty computers simultaneously. Thousands or even hundreds of thousands of computers are needed to accomplish it. This is

possible using a botnet, managed from one or more C&Cs (figure 2.2). After all, there is no ordinary computer that has the capacity and processing power to perform such an attack. It is easier and achievable, though, to infest zombie computers that will dedicate a small percentage of their processing power. Of course, the owners of the infected computers are not aware that their system is compromised.

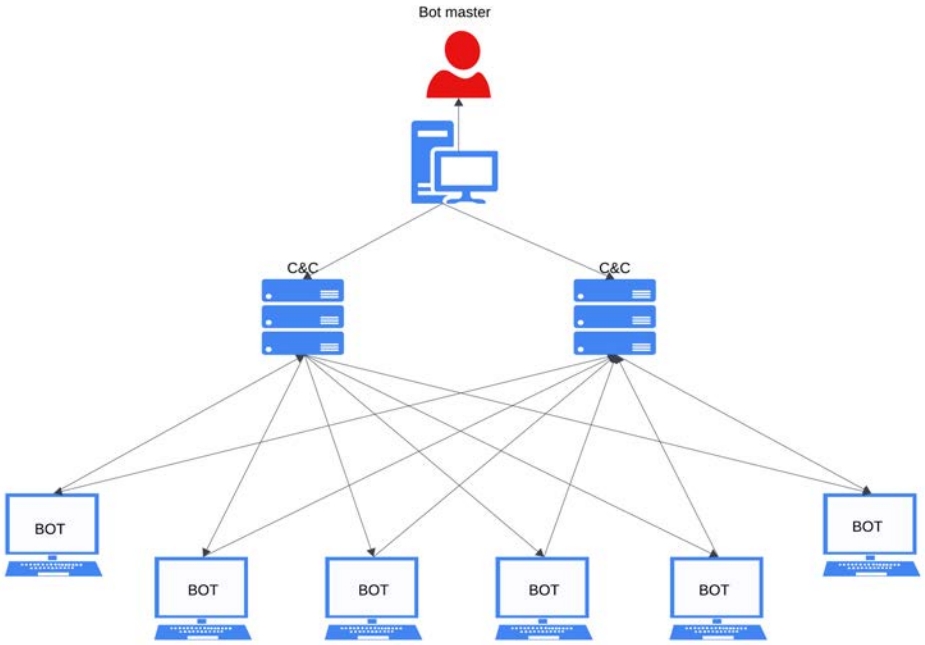

Figure 2.2: Topology of a Botnet

LDAP DDoS Attacks

Similar to the above category, in an LDAP DDoS attack, the attacker sends small queries to an exposed LDAP server, spoofing the IP address of the intended victim. These queries trigger the server to return large responses, which are again redirected to the victim. Such attacks can cause widespread disruptions, particularly if the targeted network relies heavily on directory services for authentication or authorization.

MSSQL DDoS Attacks

Microsoft SQL Server (MSSQL) DDoS attacks, like the rest of the attacks mentioned in this section, exploit vulnerabilities or misconfigurations in MSSQL servers, whose services are by default listening to ports *1433, 1434, 4022* and *135*. MSSQL servers are widely used in enterprise environments to manage databases, making them attractive targets for attackers. In

a typical MSSQL DDoS attack, attackers may exploit open or misconfigured database servers to launch reflection-based amplification attacks. They send spoofed requests to the MSSQL server, prompting it to return large responses to the victim's IP address, thereby flooding the target's network.

DNS DDoS Attacks

The role of DNS is translating human-readable domain names into IP addresses. If the DNS service is unavailable, the victim device is not able to retrieve any IP address from the DNS-held catalogue, practically rendering it useless, unless the user knows the IP addresses by heart. DNS requests use TCP/UDP on port 53.

A DNS flood attack will involve a botnet swarming with DNS queries directly to a target's DNS server, overloading its capacity and preventing legitimate users from resolving domain names. Such attacks can disrupt the availability of web services, email, and other internet-dependent systems. The following example 2.3 demonstrates an attack.

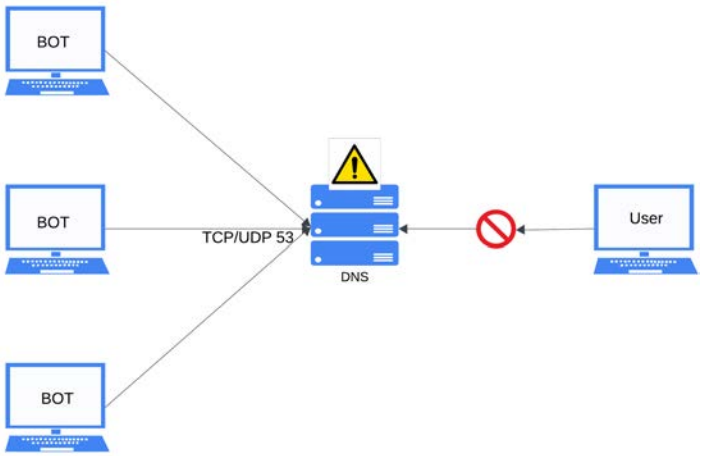

Figure 2.3: DNS flood attack

In a DNS amplification attack, each bot will use the previously described technique to send requests to DNS resolvers using spoof IP addresses that have been altered to match the actual source IP address of the intended victim. The target will then receive a response from the DNS resolvers. The attacker structures the request to produce the largest response from the DNS resolvers in order to generate an excessive amount of traffic. Consequently, the attacker's initial traffic is amplified and sent to the target, clogging their network with the bogus traffic and resulting in a denial-of-service attack. Figure 2.4 gives an example of such

an attack.

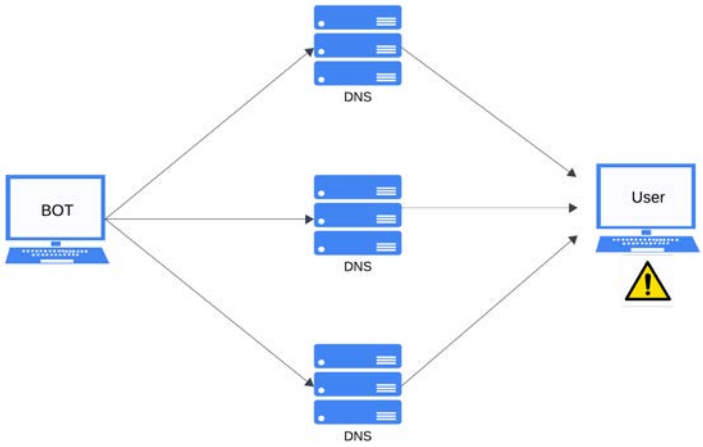

Figure 2.4: DNS amplification attack

NTP DDoS Attacks

Network Time Protocol (NTP) DDoS attacks exploit the NTP protocol *(UDP port 123)*, which is utilized for the synchronization of clocks across all devices over a network. In an NTP amplification attack, the assailant sends a small spoofed request to an NTP server making use of the IP address of the target. The server responds with a significantly larger payload, directing this amplified response to the victim and overwhelming its bandwidth.

DoS attack - TCP SYN Flood attack

The TCP protocol is typically used when two devices wish to interact in order to guarantee that the message reaches both of them. The TCP three-way handshake is the typical process. Data is permitted to flow between the two devices only after the handshake is finished. A continuous stream of SYN packets is sent to the target by the attacker in a TCP SYN flood attack. The victim must respond with a SYN/ACK packet for each SYN packet it receives. This SYN/ACK packet would be received by the attacker, who would not reply, leaving numerous half-open connections on the victim's computer.Keep in mind that the attacker is spamming TCP SYN packets continuously, which will at some point deplete the victim's computer's resources and prevent it from connecting to any other legitimate devices in the future before the attack ceases. Figure 2.5 showcases the basis of such an attack.

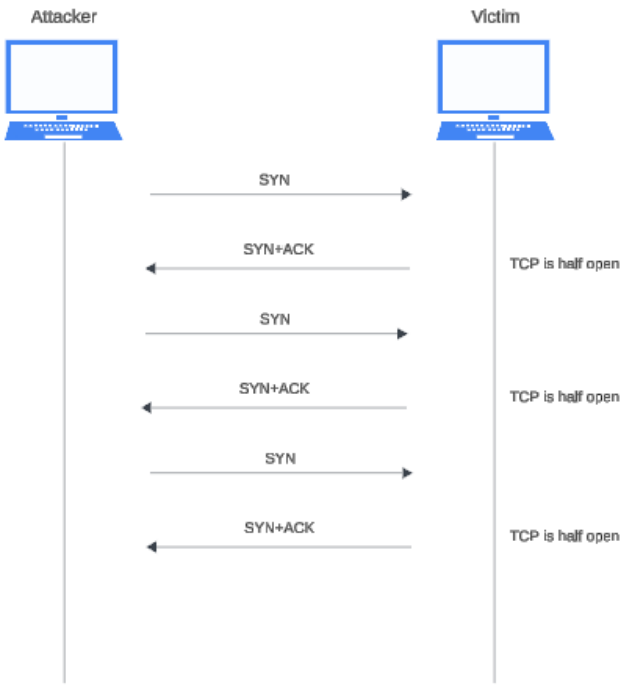

Figure 2.5: TCP SYN attack

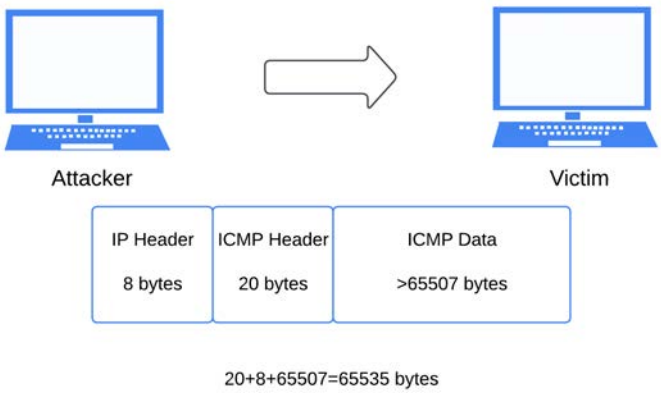

Figure 2.6: Ping of death attack

DoS attack - Ping of death

Testing of reachability between two devices can be verified using the Internet Control Message Protocol (ICMP). Attackers have the ability to alter the ICMP message's size to exceed its typical size. By default, ICMP protocol supports up to 65,536 bytes. Ping is a straightforward tool that makes use of the ICMP. The victim's computer will crash if an attacker pings with more than 65,536 bytes to another device connected to the network. This, however is almost always regulated by the system MTU, which is usually set to 1500, 1492 or 1458, limiting the maximum payload of bytes. Attacks of this type are Pings of Death.

According to the diagram 2.6, the victim machine who receives the fragmented packets will reassemble them only to discover that the total size of the packet exceeds 65,536 bytes. The system malfunctions or crashes because it doesn't know how to handle the packet, making it impossible for legitimate and authorized users to access the system. For example, the following command would crash our *e-ce.uth* server if the operation system allowed the execution of:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

c:\User\anon>ping 194.177.204.124 -t -l 65500
```

Chapter 3

Intrusion Detection Systems

3.1 Intrusion Detection Principles

Intrusion detection involves overseeing the activities happening within a computer system or network and evaluating them for indications of potential incidents, which include breaches or imminent risks of breaches of computer security policies, acceptable use policies, or standard security measures. Intrusion detection systems (IDS) mainly concentrate on detecting these events, recording details about them, and notifying security administrators. Moreover, organizations utilize IDSs for various reasons, including pinpointing issues with security policies, recording current threats, and discouraging individuals from breaching security regulations. IDSs have become an essential component of the security framework for almost every organization. [3]

3.2 IDS Detection Methods

IDS can be classified into various groups based on their deployment and detection methodologies. The two primary detection methodologies in IDS are Signature-based detection and Anomaly-based detection. Both have distinct approaches to identifying potential threats, each with its own advantages and limitations. Sections 3.2.1 through 3.2.3 will elaborate on the primary categories of detection methodologies which in the literature are generally accepted as signature-based, anomaly-based, and hybrid methodologies. It is optimal that most IDS systems make use of more than one detection methodologies, either integrated or separately, to provide more broad and accurate detection.

3.2.1 Signature-based Systems

Signature-based detection or misuse detection relies on predefined patterns or ”signatures” of known attacks. These signatures are saved in a database and used to compare with incoming, live network traffic or system events. When the IDS identifies an activity that matches a known signature, it generates an alert, indicating a potential intrusion. This method is highly effective at detecting well-known threats and specific attack patterns, such as viruses, malware, and exploitation attempts.

However, signature-based systems have limitations, particularly in their ability to detect new or zero-day attacks, which lack previously recorded signatures. Because these systems rely entirely on predefined patterns, any novel or modified threat that does not match existing signatures will likely evade detection [4]. To address this, signature databases must be regularly updated, but the process often lags behind the rapid pace of new threat emergence, creating a vulnerability window between when a threat is identified and when a signature is developed [5].

Another significant challenge of signature-based IDS is that each signature relies on an entry in a database, which in turn translates to a complete database potentially containing thousands of entries. All the entries in the database are to be compared to each arriving packet. This mechanism may be quite resource-intensive and will reduce the throughput while it’s running. Certain IDS evasion tools exploit this vulnerability by overwhelming signature-based IDS systems with an excessive number of packets (similar to DoS attacks), which causes the IDS to be unable to manage the traffic, resulting in timeouts and packet drops. As a result, this increases the chances of malicious packets avoiding inspection and being recognized as legitimate [6].

3.2.2 Anomaly-based Systems

Anomaly-based detection, on the other hand, aims to identify deviations from normal behavior. This methodology relies on building a model of normal activity for a system or network, often based on statistical methods or machine learning algorithms. Any activity that falls outside the defined ”normal” parameters is flagged as suspicious and potentially harmful.

The power of anomaly-based detection is its capability to identify new or unfamiliar threats, encompassing zero-day attacks. By continuously learning and adapting to the normal

behavior of a system, anomaly-based IDS can identify subtle changes or abnormal patterns that may indicate an attack.

However, anomaly-based detection is prone to generating false positives since normal behavior can vary over time, leading to benign activities being flagged as malicious. Tuning these systems to reduce false alarms while maintaining sensitivity to actual threats can be challenging.

3.2.3 Hybrid Systems

These systems combine the accuracy of signature-based methods with the flexibility of anomaly-based detection. In this hybrid approach, the very best technologies from both above methods can be combined, making known threats become quickly identified and neutralized through signature-based detection, while maintaining the anomaly-based component for new or sophisticated attacks.

By integrating both detection methods, hybrid IDS can be quite versatile and provide comprehensive protection against a variety of threats, reducing the likelihood of both missed attacks and false alarms. However, hybrid systems are more complex to implement and may require more resources for operation and maintenance.

3.3 Types of IDS

There are many classifications, types and subtypes of IDS. A brief description of the most common ones is provided below.

3.3.1 Network Intrusion Detection System (NIDS)

Network intrusion detection systems (NIDS) observe all traffic coming to and from devices within the network. It is positioned throughout the infrastructure at key locations, like the most critical/vulnerable subnets. Consequently, it monitors the traffic flowing through the whole subnet, making assessments based on packet data and metadata while comparing it to a database of recognized attacks. Frequently, data are also transmitted online to be analyzed by SOC teams.

NIDS can be packet-based, flow-based or session-based. Packet-based NIDS make extensive use of packet parsing and payload. That means inspecting binary data such as IP,

port, header, etc. For flow-based, the total of packets exchanged between two endpoints is scanned, so bilateral traffic inspection is involved. Lastly, for session-based systems a more statistical-kind of approach is followed. They constantly analyze the statistical data, such as packet analogies and number of packets, and also examine session details and high-level characteristics such as Server IP, Client IP, Gateway and protocol is expected. These systems are useful for preventing tunnel attacks and Trojans [7]

3.3.2 Host Intrusion Detection System (HIDS)

Host intrusion detection systems (HIDS) operate on particular endpoints (hosts) within the network. A HIDS observes only the packets that are both entering and leaving the device, and that device only. Simultaneously, it captures periodical snapshots of current system files and contrasts it with the earlier snapshots. In practice, this means that it safeguards the endpoint from both external and internal threats, examining traffic and recording harmful activity prior to alerting the administrator for further investigation. A Host-based IDS has the capacity to examine Process IDs (PIDs) and Kernell calls [8].

The techniques they use to analyze the data can be either short sequence-based or frequency-based. In the former, various system calls from a specific process are examined in order to look for anomalies that do not match its normal operation. HMMs and N-grams can belong to this category. In the case of frequency-based, the periodicity of the system calls are analyzed and how how much that deviates from standard operation. Examples in this category are Gaussian Mixture Models (GMMs) and Support Vector Data Description (SVDD).

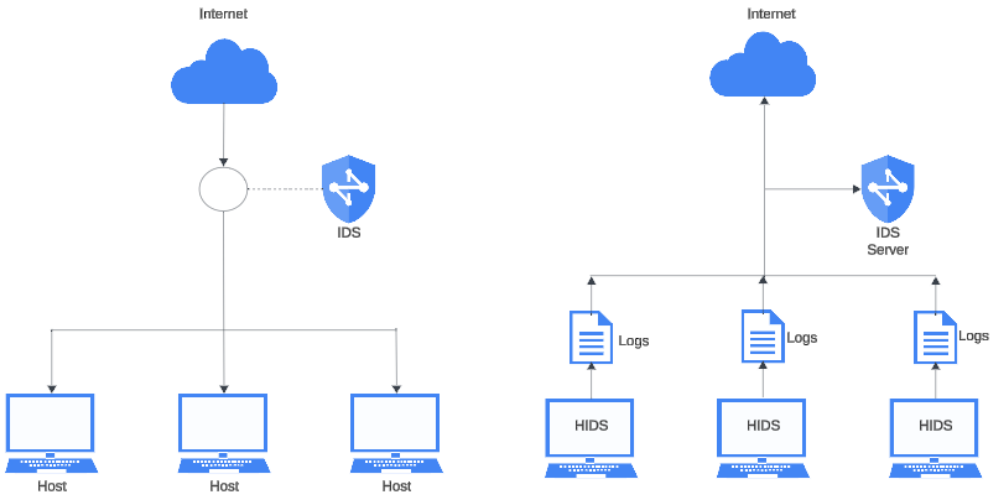

Figure 3.1: NIDS and HIDS

3.3.3 Protocol-Based (PIDS)

They comprise an agent or system that is frequently found on the front side of a service, controlling and monitoring a specific protocol in use. An example could be an HTTP-PIDS, which would analyze streams of traffic between the server and the hosts.

3.3.4 Application protocol-based (APIDS)

An APIDS comprises an agent or system that is usually found within the server side. It monitors and analyzes legitimacy of application communications between hosts and the servers regarding a specific protocol. For instance, this could be monitoring of proper SQL parsing and requests of a host to the server's database in order to evade possible SQL injection attacks. Technically, this type roughly translates as application security IDS.

3.4 Intrusion Detection System Architecture

The next section concentrates on the rudimentary system design of a network-oriented intrusion detection system. Due to differing requirements regarding the collection of data and its analysis, it is not possible to fully rely on a common standard when implementing an intrusion detection system. Nevertheless, almost all of them have some basic components/-modules that they all share [9]. These components are:

- Data gathering: The collection of data is conducted with various sensors that monitor a particular application or protocol. A module utilizing pre-processing capabilities can also be added that will carry out fundamental classification of the type of data obtained from the source.

- Detection engine: A component that executes the comparison between the collected information and the established rule-sets and triggers alerts if a discrepancy is detected.

- Database(Knowledgebase): Is a storage component that holds the rule-sets or the identifiers that the detector utilizes to compare the incoming data to.

- Output (Response): Appropriate measures are implemented following an alarm. This might be an active reaction in which the IDS executes a specified action like dropping

the packet, or a passive reaction like logging for future review by a human element to assess the suitable response.

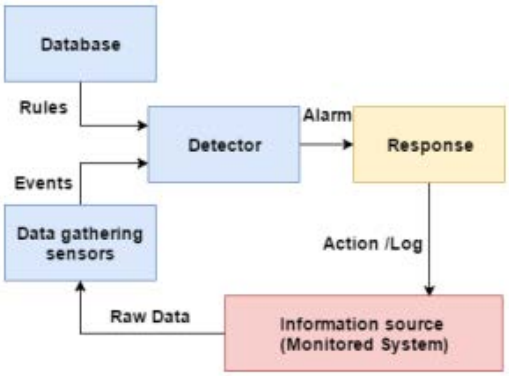

Figure 3.2: IDS Architecture[10]

3.5 IDS evasion techniques

This section covers methods including encryption, obfuscation, flooding, and fragmentation that a cyber attacker might employ to evade detection by intrusion detection systems. These strategies avoid current detection technologies, which presents a problem for IDS designers.

3.5.1 Fragmentation

The principle behind fragmentation-based evasion lies in splitting malicious payloads into smaller packets that are transmitted individually [11]. This process of fragmentation is typically handled at the network layer of the OSI, where large data packets are split into smaller fragments to adhere to the Maximum Transmission Unit (MTU) size of the network. When analyzed independently, these packets appear benign, thus deceiving IDS mechanisms that cannot reassemble packets. This technique is particularly effective against older or improperly configured IDS implementations. Many legacy IDS systems are designed to analyze packets as they arrive without reconstructing fragmented payloads.

Modern IDS solutions attempt to counter fragmentation-based evasion through packet reassembly mechanisms. These systems reconstruct the fragmented payloads into their original form before analyzing the traffic. The restructuring of packets needs the detector to hold the

data in memory and match the traffic against a signature database [12]. However, reassembly can be computationally expensive and challenging in high-traffic environments, where processing delays could lead to performance bottlenecks. Additionally, attackers may deliberately introduce delays between fragments or use overlapping fragments with conflicting data to confuse the reassembly process. Some systems might discard such ambiguous packets, while others may make incorrect assumptions during reconstruction, thereby allowing the attacker to bypass detection.

3.5.2 Encryption and Obfuscation

Encryption relies on encoding data on behalf of the attacker to make it unreadable to the IDS, while obfuscation employs techniques to disguise the structure or intent of the payload. Malicious payloads are often embedded within encrypted HTTPS traffic, preventing IDS from inspecting the packet contents. In other cases, traffic is encrypted using SSL/TLS or custom encryption. Other encryption algorithms include RSA, AES, Triple DES. Encryption ensures that only the intended recipient, such as the attacker's server, can decrypt and view the payload. Traditional IDS systems that lack the ability to decrypt traffic are rendered blind to any suspicious activity contained within encrypted communications.

Obfuscation, on the other hand, manipulates the payload or communication patterns to evade IDS detection. Attackers often encode payloads in uncommon formats or use complex transformations to avoid matching known attack signatures. For example, %3Cscript%3Ealert %28%27xss%27%29%3C%2Fscript%3E could indicate a cross-site scripting (XSS) attack payload like <script>alert('xss')</script> where the use of URL encoding disguises the original intent [11].

Defending against encryption and obfuscation-based evasion techniques poses significant challenges for IDS designers. One solution is the integration of Deep Packet Inspection (DPI) technologies, as shown in figure 3.3 which can inspect packet contents even within encrypted traffic. DPI systems work in conjunction with SSL/TLS decryption capabilities, although this requires access to encryption keys or intermediary decryption proxies [13].

Another approach involves analyzing metadata, such as packet headers, traffic flow patterns, or statistical anomalies, to identify suspicious encrypted connections. For example, detecting an unusually high volume of data being transmitted over an HTTPS connection to an unknown or untrusted server could indicate data exfiltration or malware activity, even if

the actual payload remains encrypted.

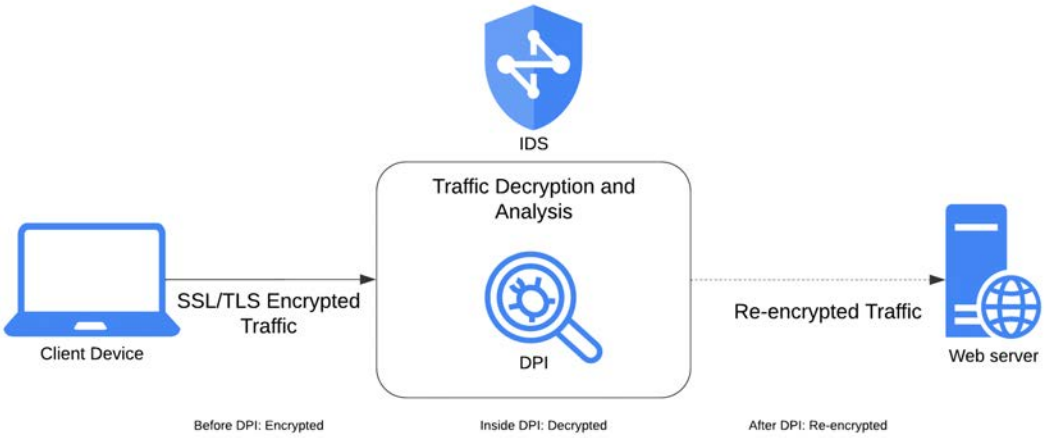

Figure 3.3: DPI usage

3.5.3 Protocol Manipulation

Protocol manipulation is a sophisticated technique used by attackers to evade Intrusion Detection Systems (IDS) by exploiting ambiguities or inconsistencies in how protocols are implemented. This technique involves crafting traffic that adheres to unusual or non-standard protocol behaviors. Attackers can bypass detection systems that fail to fully understand or analyze these deviations. This method leverages gaps in protocol parsing to ensure that malicious payloads go undetected by the IDS while being interpreted correctly by the target system.

At its core, protocol manipulation relies on the fact that many IDS implementations make assumptions about how protocols are structured and used. One common protocol manipulation technique involves TCP segmentation, where overlapping TCP segments are sent in such a way that the IDS and the target system reassemble the data differently. For example, an attacker could send two TCP segments containing overlapping data but with differing content in the overlapping portion. The IDS might assume one version of the data during analysis, while the target system processes the other version.

Attackers also manipulate DNS protocols, embedding malicious payloads in DNS queries or responses. Since DNS traffic is often considered benign and overlooked by many IDS solutions, attackers may use this as a covert channel for data exfiltration or command-and-control (C2) communication. For example, the data exfiltration payload might be encoded

in the subdomains of a DNS query, such as my.name.is.dimitris.uth.gr Without advanced inspection capabilities, the IDS might fail to detect this behavior.

3.5.4 Traffic Flooding

Traffic flooding is a widely used evasion technique where attackers overwhelm an Intrusion Detection System (IDS) by generating a large volume of traffic, effectively masking malicious activity within the flood. Given the limited processing capacity of IDS systems, this forces them to prioritize speed over quality analysis. This strategy can lead to the IDS either missing the malicious payloads or being knocked out entirely due to resource exhaustion.

A classic example of traffic flooding is the use of Distributed Denial-of-Service (DDoS) attacks, as discussed in section 2.3.3, where attackers leverage botnets to generate massive amounts of network traffic. An attacker could simultaneously conduct a secondary attack, such as data exfiltration or malware delivery, which may escape detection [14]. An attacker might also generate legitimate-looking packets with randomized but irrelevant payloads to trigger false positives. These false alarms desensitize the IDS and its human operators, who may begin ignoring alerts due to alert fatigue. This ”signal-to-noise” approach ensures that real attacks are buried under a heap of irrelevant traffic, delaying or entirely preventing detection [15]. Also dimensions of traffic flooding constitute the previously discussed *Ping of Death* and *TCP Flood* techniques in sections 2.3.3 and 2.3.3 respectively.

Chapter 4

Implementation

4.1 Performance metrics

Intrusion Detection Systems can be evaluated in a number of ways for efficiency and effectiveness based on evaluation datasets. Different metrics have been suggested over time to evaluate different features of an IDS, such as performance, usability or correctness. Test datasets are utilized to assess the performance of a machine learning algorithm after it has been trained [10]. Tasks in machine learning can be divided into classifications or regressions. Evaluating various machine learning tasks may be accomplished using various performance metrics. That is, for an algorithm to be effective, it needs to be somehow assessed. Metrics utilized to assess the model's generalization capability are significant when handling newly obtained datasets. When assessing a model, it's essential to consider additional metrics apart from classification accuracy to assess its effectiveness. The performance metrics that are utilized in this thesis are explained in what follows.

4.1.1 Confusion matrix

In situations where a model has been trained to classify detected network traffic in two or more classes (e.g. Benign vs malicious) consists of two or more types of classes, the easiest and most straightforward way to assess its classification performance is by using a confusion matrix. A confusion matrix can be used to assess model accuracy and reliability. To compute the confusion matrix in the study conducted in this thesis, four crucial scenarios have been considered.

1. True positive (TP): Intrusions that the IDS successfully detects are known as true positives.
2. False positive (FP): When an IDS incorrectly classifies normal or non-intrusive behavior as intrusive.
3. True Negative (TN): Typical behavior that the IDS successfully classifies as normal or benign.
4. False Negative (FN): Intrusions missed by the IDS and classified as normal or non-intrusive are known as false negatives.

Actual	**Predicted**	
	Attack	**Normal**
Attack	TP	FN
Normal	FP	TN

Table 4.1: Confusion matrix

4.1.2 Accuracy or Classification rate (CR)

It is defined as the ratio of correctly classified instances to the total number of instances, i.e., the percentage of accurate predictions made by the model, and is likely the most recognized and easily comprehensible evaluation metric.

$$CR = \frac{TP + TN}{TP + FP + TN + FN} \tag{4.1}$$

High accuracy means the model is generally predicting the correct labels, but it doesn't account for class imbalances. Class imbalance occurs when some classes in a dataset are significantly more frequent than others, which can lead to misleading performance metrics. For example, in heavily biased and imbalanced datasets, a model that predicts only the majority class will have high accuracy but poor performance overall. In the context of Intrusion Detection, if benign traffic vastly outnumbers attack types, a model that predicts all traffic as benign might still have high accuracy but would be ineffective at detecting actual attacks.

4.1.3 False positive rate (FPR)

It is the ratio of the total number of normal instances to the number of normal instances that were identified as attacks.

$$FPR = \frac{FP}{FP + TN} \tag{4.2}$$

A false alarm occurs when a legitimate packet is mistakenly identified as harmful, leading to the IDS unnecessarily triggering the system's response. An ideal model would exhibit no false positives, resulting in a FPR of 0.0, meaning a 0% false alarm rate. FPR is a component of the ROC curve, further discussed later.

4.1.4 Precision (PR)

Precision is considered as the fraction of data instances predicted as positive that are actually positive.

$$PR = \frac{TP}{TP + FP} \tag{4.3}$$

High precision indicates that when a model predicts a positive class, it is very likely correct. A high precision means fewer false alarms. A perfect, ideal model would have a precision of 1.0 and zero false positives.

4.1.5 Recall, Sensitivity, or True Positive Rate (TPR)

This is the proportion of correctly identified positive samples, or more simply, the ratio of correct positive predictions to the total positive samples.

$$Recall = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{4.4}$$

Precision increases as false positives reduce, while recall improves when false negatives are reduced. Consequently, precision and recall frequently exhibit an inverse relationship, so that enhancing one diminishes the other. High recall means the model is very good at finding all the positive instances. Missing a positive instance has serious consequences, in this case failing to detect an actual intrusion.

4.1.6 F1-Score

F1-score is a metric which involves both recall(TPR) and precision(PR) and is defined as:

$$F_1 = 2 \cdot \frac{PR \cdot TPR}{PR + TPR} \tag{4.5}$$

The F1 score balances precision and recall, making it useful when a single metric is needed for a model on an imbalanced dataset. A high F1 score means the model is balanced between false positives and false negatives.

4.1.7 AUC-ROC Curve

The AUC-ROC Curve (Area Under the Receiver Operating Characteristic Curve) is another critical evaluation metric used for binary classifiers, which includes IDS models that classify network traffic as either *Benign* or *Non-Benign*. AUC-ROC practically measures the ability of the model to distinguish between classes. It incorporates plotting the True Positive Rate (eq.4.4) against the False Positive Rate (eq.4.2) at various thresholds, and is especially useful for evaluating models on imbalanced datasets. Better performance is indicated by a curve that approaches the upper-left corner. The top-left corner is the ideal situation because it maximizes the true positive rate while minimizing the false positive rate.

For an IDS, a high AUC signifies that the model is good at distinguishing between normal traffic and attacks. Even if the dataset is imbalanced, the AUC metric helps evaluate the overall classification performance regardless of the threshold used to classify traffic as an attack or normal.

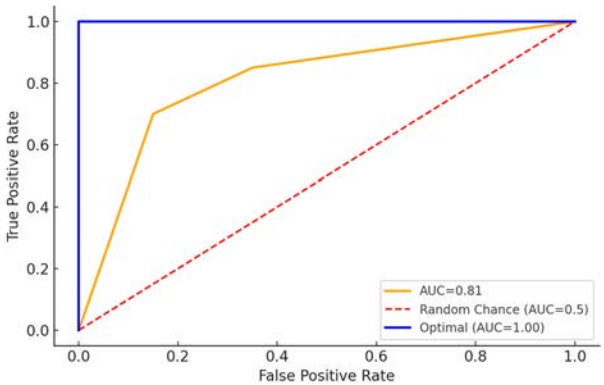

Figure 4.1: An example of AUC & ROC Curve

The ROC curve is summarized by a single value known as the AUC (Area Under the Curve) score. It shows the likelihood that a randomly selected attack instance will be ranked higher by the model than a randomly selected normal instance. The range of the AUC score is 0 to 1. A model with a discriminative ability of 0.5 is equivalent to random guessing, whereas a model with a classification of 1 is perfect (perfect separation between normal and attack). Figure 4.1 illustrates the above.

4.1.8 Pearson Correlation Coefficient

This formula measures the linear correlation between two variables, providing a value between -1 and 1.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

This coefficient is useful in statistics and data analysis to understand how changes in one variable are associated with changes in another, which helps in identifying patterns or predicting one variable based on the other.

4.2 Datasets

High-quality datasets are crucial for the development and evaluation of intrusion detection systems, as they form the basis for testing and training machine learning models. A number of datasets have been used in the IDS domain, from legacy synthetic data to state-of-the-art real-world traffic. This overview will look at some of the most used datasets in IDS research, considering their features, strengths, and limitations.

The DARPA 1998 and 1999 datasets, prepared by the MIT Lincoln Laboratory, are among the earliest and most widely used datasets in IDS research. These datasets simulate network traffic in a military environment and contain both normal behavior and a variety of attack scenarios, including Denial of Service (DoS), User-to-Root (U2R), Remote-to-Local (R2L), and Probes [16]. The DARPA 1998 dataset consists of nine weeks of data, with seven weeks allocated for training and two for testing, while the DARPA 1999 dataset introduces additional attack types. Despite their historical significance, these datasets face criticism for their outdated protocols, lack of real-world realism, and simulated nature. Nevertheless, they were instrumental in early benchmarking efforts, especially for classical machine learning models like decision trees and neural networks [17].

A derivative of the DARPA dataset is the KDD Cup 1999 dataset, which became the benchmark for IDS research after its use in a machine learning competition. It features 41 attributes derived from TCP connections, covering the same attack categories as DARPA. However, KDD Cup 1999 has significant flaws, including redundant and duplicate records that artificially inflate the performance of ML models and unrealistic attack-to-normal ratios [18]. Despite these limitations, due to its availability and easy access, it has become a standard to evaluate clustering, ensemble learning, and other ML techniques.

To overcome the drawbacks in KDD Cup 1999, the NSL-KDD dataset was proposed. This enhanced version eliminates duplicate records, provides balanced class distribution, and presents a more realistic evaluation framework for IDS models [18]. NSL-KDD is still in use for benchmarking ML techniques such as SVMs and random forests due to its corrected flaws. However, its relatively small size limits its applicability to modern large-scale network environments.

The UNSW-NB15 dataset, created by the University of New South Wales, further bridges the gap between outdated datasets and modern network requirements. It contains 49 features derived from network flows and includes nine attack categories, such as worms, backdoors

and exploits. UNSW-NB15 is particularly valuable for hybrid IDS evaluations, as it balances attack representation while keeping the size reasonable for ML processing [19]. Nevertheless, it is much smaller in comparison with CICIDS2017, which may affect scalability.

The UWF-NB15 dataset extends the above mentioned UNSW-NB15 dataset by inclusion of modern patterns of traffic flow and updated new forms of cyberattacks. It maintains a perfect balance of attack and normal traffic for UWF-NB15, that provides a valid outcome for assessing machine learning models. With its various types of attack classifications and vast dataset size, it offers diversified applications and hybrid traditional IDS frameworks over the challenges of security on ever-varying cyberterrors [19].

Real datasets like MAWILab, coming from the MAWI archive, provide another perspective as they capture large-scale anonymized internet traffic. Unlike generated datasets, MAWILab is aimed at finding anomalies in real traffic without any predefined labels indicating attacks [20]. This dataset is perfect for unsupervised and semi-supervised learning approaches, as the unavailability of labeled attack data renders the applicability of supervised ML techniques difficult.

Datasets related to IoT environments, like TON_IoT and Bot-IoT [21], are becoming highly relevant with the increasing usage of IoT devices. TON_IoT combines network traffic, telemetry, and application logs to simulate modern IoT-specific attacks such as data injection and DDoS. Similarly, Bot-IoT focuses on IoT vulnerabilities on a massive scale of more than 72 million records. These datasets are indispensable for IoT-specific intrusion detection, especially for deep learning and multimodal analysis.

The paper by Sharafaldin, Lashkari, and Ghorbani (2019) [22], provides several modern datasets for intrusion detection, focusing on realistic network scenarios and contemporary attack methods. Among the most significant datasets introduced in this work is CSE-CIC-IDS2018, which simulates modern enterprise network traffic, including both normal and malicious activities. The dataset was developed to overcome some of the shortcomings of older datasets in terms of providing realistic traffic patterns, up-to-date attack types, and comprehensive labeling. It contains brute-force scenarios such as SSH and FTP, SQL injection, cross-site scripting (XSS), DDoS, and botnet, more than 16 GB of raw network traffic with extracted flow features. In particular, this dataset will be interesting for the evaluation of anomaly-based, misuse-based, and hybrid detection systems, making it one of the most complete resources available for machine learning research in intrusion detection. Another

dataset introduced here is CIC-DDoS2019, developed to specifically target Distributed Denial of Service (DDoS) attacks. This reflects the prevalence of DDoS threats nowadays and captures 12 different types of such attacks, such as TCP-SYN flood, UDP flood, and HTTP flood. Due to its focus on flow-based statistical features extracted from network traffic, CICDDoS2019 is an important contribution to testing machine learning models for the detection and mitigation of large-scale distributed attacks. This effort also produced the CICIDS2017 dataset, which remains one of the most widely used datasets in intrusion detection research. It provides seven days of network traffic, including both normal activities and malicious attacks such as brute force, web attacks, DoS, Heartbleed, botnets, and infiltration. CICIDS2017 is characterized by its detailed labeling of traffic flows based on over 80 attributes and its simulation of realistic user profiles within a network environment. This dataset is widely used as a benchmark for testing and comparing machine learning and deep learning approaches to intrusion detection. Besides the datasets mentioned above, the paper [22] presents the CIC-AWS-2018, resource focused on cloud security. This dataset was generated using Amazon Web Services (AWS) and captures a wide variety of normal and attack traffic in a cloud infrastructure. It includes scenarios such as DoS, DDoS, brute force, and infiltration targeting cloud-hosted virtual machines. CIC-AWS-2018 offers a realistic view of security challenges in cloud platforms and is particularly relevant for developing intrusion detection systems tailored to cloud computing environments.

The CIRA-CIC-DoHBrw-2020 dataset [23] is focused on the identification of malicious behavior in DNS-over-HTTPS (DoH) traffic. With the emergence of DoH, which encrypts DNS queries to improve privacy, traditional inspection techniques have become increasingly complex. This dataset is of particular value for testing machine learning methods for detecting encrypted tunneling and other DoH-based threats. It includes a mix of legitimate and malicious encrypted DNS traffic, supporting advanced anomaly detection and traffic classification models.

The application-layer attack CIDDS-001 dataset, developed within the Coburg Intrusion Detection Data Sets project, covers both normal and malicious traffic in a simulated network environment, with a focus on Layer 7 activities such as HTTP and FTP-based attacks. The CIDDS-001 is very effective in hybrid intrusion detection systems that combine the principles of misuse and anomaly detection. Its structured approach to capturing application-layer anomalies makes it a critical resource for modern IDS research [24].

The ISCX-IDS2012 dataset, generated by the Canadian Institute for Cybersecurity, is considered one of the most important web application attack research datasets. It contains six main attack categories, including SQL Injection and Brute Force, along with normal traffic to simulate a realistic network traffic environment. This dataset is especially useful for evaluating signature-based and anomaly-based IDS approaches because it reflects comprehensive coverage of contemporary web application vulnerabilities [25].

For cloud and encrypted network traffic, the ISCX VPN-Tor dataset [26]is particularly useful. It includes traffic generated in normal VPN environments and over the Tor network, alongside malicious activities such as port scanning and brute force attacks. This dataset addresses the challenges of analyzing encrypted traffic and has been widely adopted for research in encrypted communication anomaly detection and VPN traffic classification.

Another interesting dataset is CTU-13 [27], generated at the Czech Technical University. It is a botnet traffic analysis oriented dataset that contains 13 capture scenarios, each containing normal and botnet-infected traffic. This dataset focuses on network flow characteristics and raw packet captures. It is very suitable for command-and-control (C&C) traffic detection and anomaly-based intrusion detection. Its rich botnet data has established CTU-13 as a benchmark for evaluating machine learning techniques in botnet detection.

Dataset	Year	Attack Types	Primary Use Case
DARPA 1998/1999	1990s	DoS, U2R, R2L, Probe	Benchmarking classical ML models
KDD 1999	1999	DoS, U2R, R2L, Probe	Clustering and classification
NSL-KDD	2009	DoS, U2R, R2L, Probe	Corrected evaluation framework
UNSW-NB15	2015	Worms, Backdoors, Exploits, Fuzzers	Hybrid IDS evaluation
MAWILab	Ongoing	Anomalies in real-world traffic	Anomaly detection
TON_IoT	2020s	IoT-specific, DDoS, Data Injection	IoT-specific IDS
Bot-IoT	2019	IoT-specific, Botnet	Botnet detection in IoT
CSE-CIC-IDS2018	2018	Brute Force, XSS, DDoS, Botnets	Enterprise network IDS
CIC-DDoS2019	2019	DDoS-specific, TCP-SYN, UDP, HTTP Flood	DDoS attack analysis
CICIDS2017	2017	Brute Force, DoS, Botnets, Heartbleed	General-purpose IDS benchmark
CIC-AWS-2018	2018	DoS, DDoS, Brute Force, Cloud Infiltration	Cloud-specific IDS
DoHBrw2021	2021	DNS-over-HTTPS tunneling	Encrypted traffic analysis
CIDDS-001	2017	HTTP, FTP-based application-layer attacks	Application-layer intrusion
UWF-NB15	2022	Updated UNSW-NB15 attack scenarios	Modern cybersecurity threats
ISCX-IDS2012	2012	SQL Injection, Brute Force	Web application security
ISCX VPN-Tor	2016	Encrypted VPN and Tor-based traffic	VPN/Tor anomaly detection
CTU-13	2014	Botnet traffic, C&C detection	Botnet and C&C detection

Table 4.2: Comparative Overview of Intrusion Detection Datasets

Chapter 5

Testing using CIC-DDoS2019

5.1 Introduction to DDoS Testing

In recent years, Distributed Denial-of-Service (DDoS) attacks have grown significantly in frequency, scale, and sophistication. As highlighted in Section 2.3.3, these attacks aim to overwhelm network resources, blocking services to legitimate users. In October 2023, Google announced that they mitigated the largest, until then, DDoS attack [28] - 398 million requests per second (RPS). The largest Distributed Denial of Service (DDoS) attack in history occurred in 2024 [29], targeting multiple industries including finance, telecommunications, and the internet sector. This unprecedented attack peaked at an astonishing 3.8 TBps.

The CIC-DDoS2019 dataset is a comprehensive and well-structured dataset specifically designed for evaluating intrusion detection systems (IDS) in the context of distributed denial-of-service (DDoS) attacks. Developed by I. Sharafaldin et al. [30], this dataset is highly regarded in the cybersecurity research community due to its realistic traffic scenarios, which include a variety of modern DDoS attack types (e.g., SYN flood, UDP flood, DNS flood) alongside benign traffic, enabling accurate testing and validation of detection models. The CIC-DDoS2019 dataset stands out due to the following reasons:

- It covers almost all known examples of DDoS attacks.
- It provides realistic traffic scenarios, simulating real-world attack conditions.
- It contains well-labeled, high quality data, including ~ 80 extracted traffic features.
- It is acknowledged and supported in academia for both anomaly-based and signature-based IDS.
- It includes botnet-based traffic, making it one of the few datasets to do so.

- It is recently published & open source.

5.2 Dataset Overview

The dataset was created using a controlled network environment, simulating attacks on a victim network over two days. The first day was dedicated to testing scenarios, while the second day focused on training data generation. Live traffic was captured using CICFlowMeter-V3, [31] a versatile open-source network traffic analysis tool developed by the Canadian Institute for Cybersecurity (CIC). It is primarily used for generating flow-based datasets by converting raw network traffic (PCAP files) into a structured format suitable for machine learning and cybersecurity research. With this tool, 80 features were extracted from network flows, including timing metrics, protocol-specific attributes and traffic volume statistics, making it particularly useful for detecting and analyzing anomalies, such as DDoS attacks. By using CICFlowMeter-V3, the researchers were able to ensure consistent, high-quality feature extraction making this tool instrumental in generating the CIC-DDoS2019 dataset. However, preprocessing steps were necessary to clean and optimize the dataset.

5.3 Cleaning

Initially, the dataset consists of 30GB of data distributed across 18 CSV Files with a total of 70,000,000 samples and 87 features. They had to ensure that the data are correctly cleaned-up, that duplicates are removed, that there are no missing (NaN) values, and that some irrelevant/non-useful features are removed to speed up the computational process. This process has successfully been tested and proposed by Agostinello et al. [32]. In figure 5.1, from [32], a more detailed summary of the datasets used for training and testing is provided.

5.3.1 Removal of metadata & single value columns

The initial 87 features included metadata columns generated by the CICFlowMeter that are completely irrelevant to the detection of DDoS attacks, such as *"Flow ID"* and *"Timestamp"*. Features *"Fwd Header Length", "Src IP", "Dst IP", "SimilarHTTP"* contain information about the local network topology and are also irrelevant so they were dropped. Additionally, features *"Src Port", "Dst Port"* were removed since all of the tested proto-

cols use well-known ports and also to avoid becoming ”shortcuts” for the model predictions. Lastly, the columns *Bwd PSH Flags*, *Fwd URG Flags*, *Bwd URG Flags*, *FIN Flag Count*, *PSH Flag Count*, *ECE Flag Count*, *Fwd Avg Bytes/Bulk*, *Fwd Avg Packets/Bulk*, *Fwd Avg Bulk Rate*, *Bwd Avg Bytes/Bulk*, *Bwd Avg Packets/Bulk*, *Bwd Avg Bulk Rate* were found to be always of zero value so they were also dropped.

5.3.2 Removal of NaN & Inf values

Rows containing *NaN*, *Infinite*, or *Null* values were removed, as described by [33] and [34]. These values can be problematic to troubleshoot later and might also affect our data integrity, hindering our model's ability to learn from the data in the long term. Moreover, if a model is trained on data containing NaN values, it can lead to harder interpretation and more ambiguous, unpredictable behavior. Following the methodology proposed by [32], [35] and depicted in more detail in figure 5.1, the total number of removed rows amounted to 92,604.

5.3.3 Duplicate rows removal

Finally, as last action the data were checked for fully duplicate rows. As a general rule, duplicates can bias training and might lead to over-optimistic estimates of classification performance during testing. Furthermore, they significantly increase the computational load needed to build our model. Duplicate rows comprised a remarkable 85.10% of the dataset, as outlined in figure 5.1.

5.4 Balancing

Our original dataset, as displayed in figure 5.1 exhibited extreme class imbalances. A quick glimpse of the final data reveals, for instance, 35,000:1 analogy in training data between classes *UDP* and *LDAP*. Analogy between *TFTP* and *NETBIOS* is an astounding 209,000:1. As stated earlier, the original CIC-DDoS2019 dataset contained 70,427,637 rows but with a heavy skew towards malicious traffic as shown in table 5.1. The benign-to-malicious ratio was adjusted to approximately 1:1.7 for the training set and 1:5 for the testing set. This involved significant undersampling of benign traffic and strategic sampling (both under and over) of malicious categories to ensure representativeness across attack types. These adjustments resulted in a two new final datasets of 125,170 rows for training and 306,201 rows

Filename	Rows(initial)	Duplicates (#)	Duplicates (%)	N. inf	N. NaN	Rows(final)
train/LDAP	2,181,542	2,150,051	98.56	2,480	10	31,491
train/MSSQL	4,524,498	4,315,627	95.38	26,715	9	208,871
train/NetBIOS	4,094,986	4,073,870	99.48	2,968	8	21,116
train/SNMP	5,161,377	5,045,899	97.76	4,135	11	115,478
train/SSDP	2,611,374	1,719,417	65.84	1,538	2	891,957
train/UDP	3,136,802	2,059,422	65.65	1,401	7	1,077,380
train/UDPLag	370,605	277,586	74.90	68	2	93,019
train/Syn	1,582,681	1,426,794	90.15	30	6	155,887
train/TFTPsv	20,107,827	15,688,113	78.02	210	22	4,419,714
train/DNS	5,074,413	4,958,122	97.71	10,146	22	116,291
train/NTP	1,217,007	90,774	7.46	225	25	1,126,233
test/Portmap	191,694	185,605	96.82	143	1	6,089
test/NetBIOS	3,455,899	3,444,578	99.67	397	3	11,321
test/LDAP	2,113,234	2,089,308	98.87	2,911	7	23,926
test/MSSQL	5,775,786	5,501,052	95.24	36,281	5	274,734
test/UDP	3,782,206	2,484,620	65.69	2,286	2	1,297,586
test/UDPLag	725,165	585,350	80.72	163	3	139,815
test/Syn	4,320,541	3,840,392	88.89	346	16	480,149
Total	70,427,637	59,936,580	85.10	92,443	161	10,491,057

Figure 5.1: Statistics of DDoS Dataset after Preprocessing.[32]

for testing, with benign and malicious traffic distributed in both. Table 5.2 summarizes the traffic type distribution in each of the two processed new datasets.

5.5 Pre-processing

After examining our final data version (Table 5.2), it is obvious that further actions are required before we are ready to build our model. The steps performed at this stage are analyzed below.

Traffic Type	# Rows	% of Total
Normal Traffic	113,828	0.16%
Malicious Traffic	70,313,809	99.84%
Total	70,427,637	100.00%

Table 5.1: Initial traffic type distribution in the CIC-DDoS2019 dataset.

Label	Count
Syn	48840
Benign	46427
UDP	18090
MSSQL	8523
LDAP	1906
Portmap	685
NetBIOS	644
UDPLag	55

Label	Count
DrDoS_NTP	121368
TFTP	98917
Benign	51404
DrDoS_UDP	10420
UDP-lag	8872
DrDoS_MSSQL	6212
DrDoS_DNS	3669
DrDoS_SNMP	2717
DrDoS_LDAP	1440
DrDoS_NetBIOS	598
Syn	533
WebDDoS	51

Table 5.2: New Testing & Training Data distribution

5.5.1 Label Mapping

Common Label mapping between the training and testing datasets was necessary in order to ensure consistency. Several of the original labels were representing variations of the same attack type (e.g., *DrDoS_UDP*, *UDP*). A predefined label mapping dictionary was used to standardize these labels into a consistent format. In our case, column names in the testing set were renamed to match those of the training set.

5.5.2 Removal of highly correlated columns

A correlation analysis was also conducted, similar to the research of Becerra et al. [36]. The heatmap in Figure 5.2 indicates some pairs of features with high collinearity. The already removed columns from chapter 5.3 are absent, creating the noticeable gaps. Multicollinearity affects the model's ability to generalize well on new data and can lead to overfitting. Essentially, highly correlated features do not provide new information to the model at all. By removing them, it is ensured that the model focuses on the most unique features. Plot 5.3

shows the number of columns with

$$\rho_{X,Y} > 0.8$$

A total of 34 highly correlated features were removed, displayed in the following vector:

['Bwd Packets Length Total', 'Fwd Packet Length Mean', 'Bwd Packet Length Mean', 'Bwd Packet Length Std', 'Flow IAT Std', 'Flow IAT Max', 'Fwd IAT Total', 'Fwd IAT Mean', 'Fwd IAT Std', 'Fwd IAT Max', 'Fwd IAT Min', 'Bwd IAT Std', 'Bwd IAT Max', 'Fwd Packets/s', 'Packet Length Min', 'Packet Length Max', 'Packet Length Mean', 'Packet Length Std', 'Packet Length Variance', 'RST Flag Count', 'Avg Packet Size', 'Avg Fwd Segment Size', 'Avg Bwd Segment Size', 'Subflow Fwd Packets', 'Subflow Fwd Bytes', 'Subflow Bwd Packets', 'Subflow Bwd Bytes', 'Fwd Act Data Packets', 'Fwd Seg Size Min', 'Active Max', 'Active Min', 'Idle Mean', 'Idle Max', 'Idle Min']

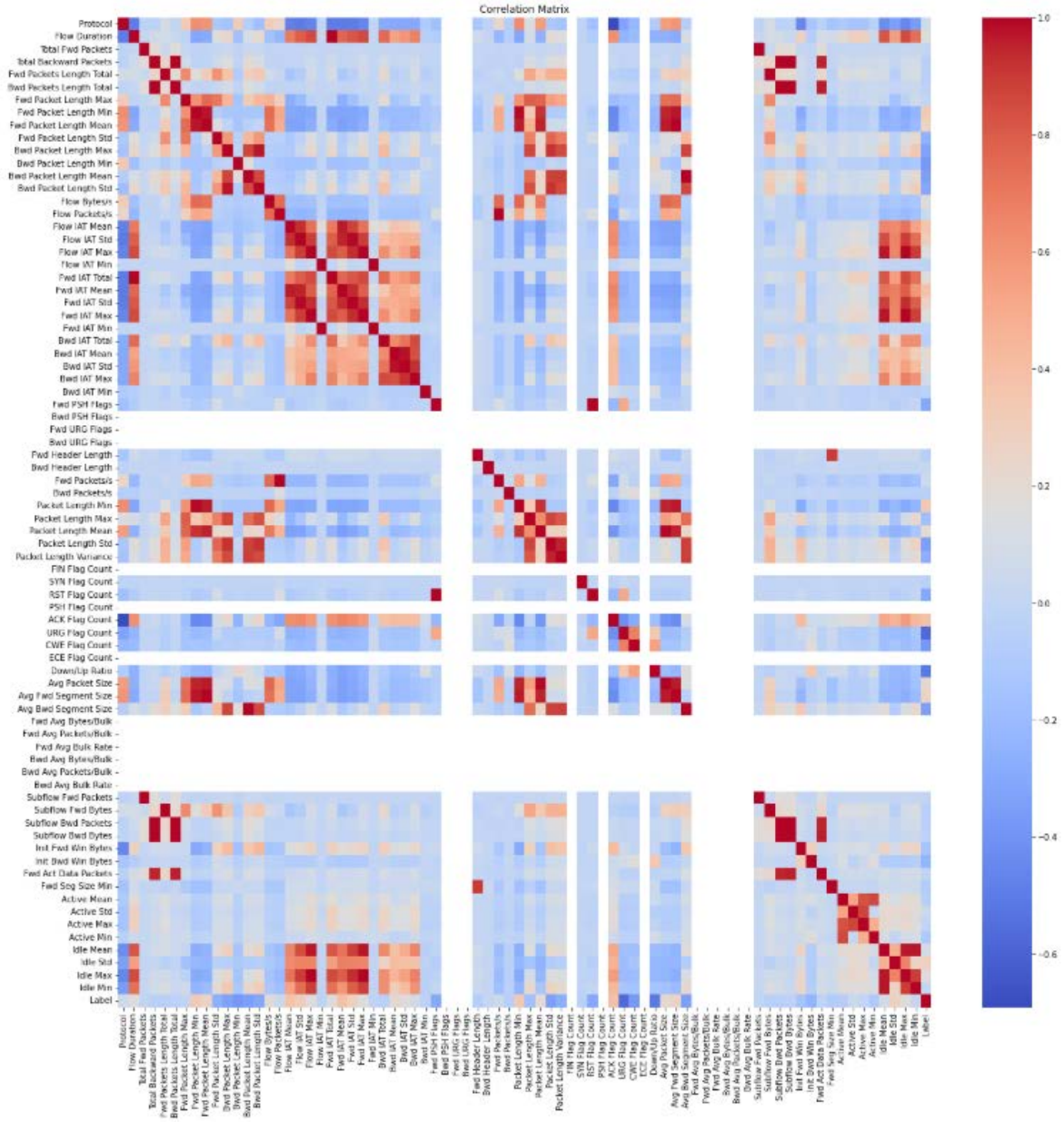

Figure 5.2: Correlation Matrix

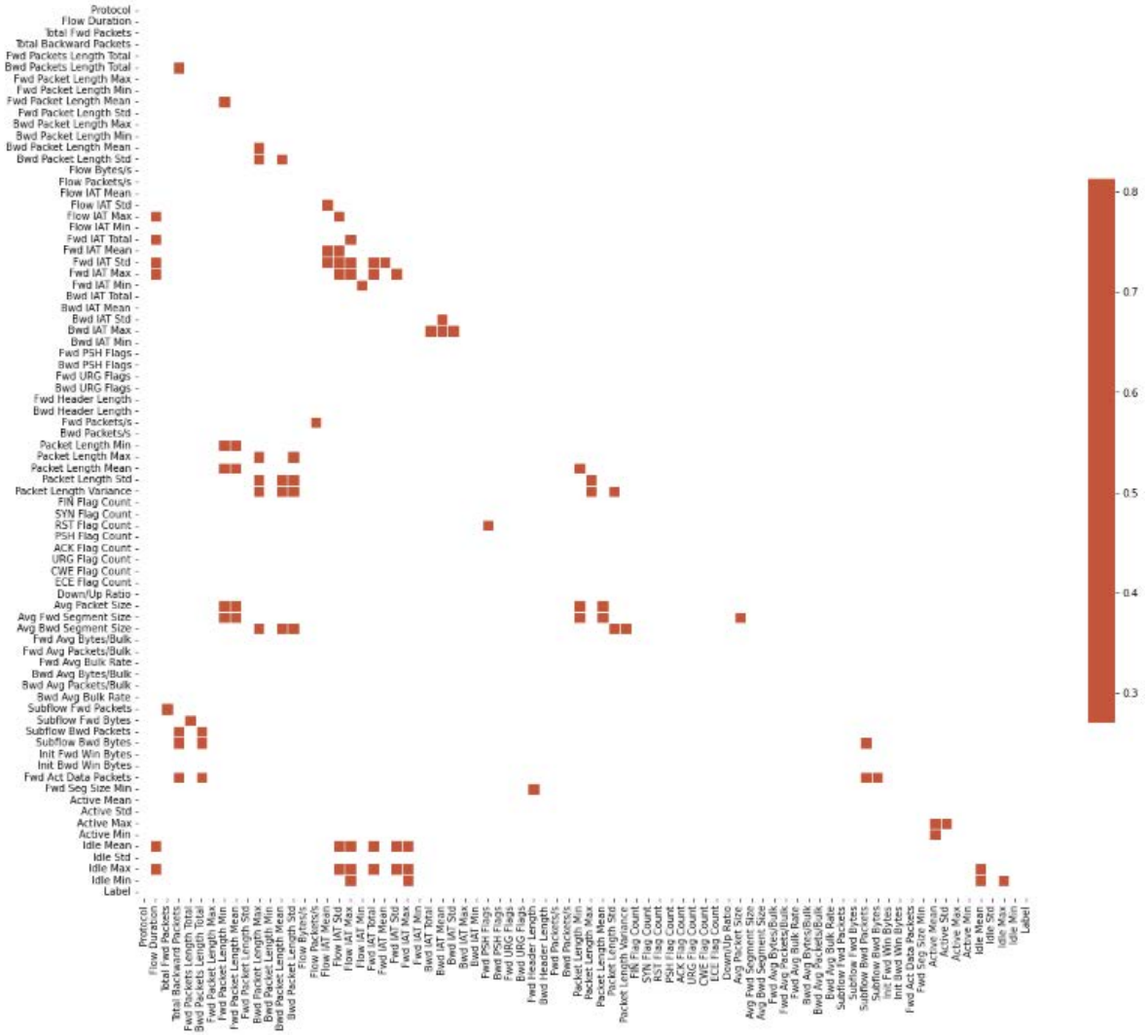

Figure 5.3: Columns with Pearson correlation coefficient above 0.8

5.5.3 Feature Categorization & Scaling

The features in the dataset were first categorized, followed by the application of scaling and transformation techniques. Each column in the dataset was classified into one of the following categories based on its characteristics, similar to [36]:

1. Categorical Features: Features with discrete, often text-based values or numeric codes representing distinct classes. Examples include protocol types, IP addresses, or flags.

2. Numerical Features: Representing quantitative data, such as packet sizes, flow durations, or byte rates.

3. High-Cardinality Categorical Features: Features with a large number of unique values, such as source or destination IP addresses. Such features often do not directly contribute

to the prediction and can significantly inflate memory and computational costs.

MinMaxScaler was applied to normalize all numerical features within the range [0, 1]. This ensured that features with varying scales did not disproportionately influence the model training process and guaranteed uniformity across the dataset. The formula for MinMax scaling is:

$$X_{\text{scaled}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

This process is more beneficial for models sensitive to feature magnitudes, such as neural networks and distance-based algorithms like the K-NN that was used in this research.

Chapter 6

Experimenting

6.1 Introduction

To thoroughly evaluate the effectiveness of machine learning models using the new dataset, three distinct experiments were conducted. Each experiment aimed to address a specific perspective of the dataset while taking advantage of the preprocessing steps described in the previous sections. The three distinct pipelines were created using Python and the *SKlearn* library as well as classic additional libraries such as *pandas* for data analysis and *numPy* for computation. The environment used was a Jupyter notebook running within a Docker container.

Experiments 1 & 2 involved multiclass classification, while Experiment 3 treated the problem as binary. In the first two experiments, various machine learning models were employed, including Random Forest, K-Nearest Neighbors (KNN), XGBoost, and Multi-Layer Perceptron (MLP). Each of these models provides different strengths in handling classification. Random Forest and KNN are classic models known for their robust handling of imbalanced data and general classification tasks, while XGBoost is particularly effective in dealing with complex patterns through its gradient boosting mechanism. MLP, being a neural network model, offers the ability to learn patterns in data, especially when the relationships between features are non-linear.

6.2 Exp. 1: Training on Mutually Represented Classes

The original datasets contain numerous traffic types, some of which are unique to a single dataset[37], [38]. For example, classes like *WebDDoS* and *DrDoS_SNMP* appear exclusively in our ”Training” dataset while on the other hand, class *Portmap* can be found exclusively in the ”Testing” data. Table 5.2 clearly indicates labels not present in both datasets.

In this first experiment, the non-common classes were dropped from the two datasets, evaluating the model's multi-class classification by utilizing the remaining classes, as depicted in the following table 6.1. The mutually represented classes included both attack and benign traffic types. The distributions, however, were not perfectly balanced, with some classes, such as *UDPLag* in the testing dataset, being significantly overrepresented relative to the same class in the training data. These imbalances were taken into account during the analysis of model performance. The primary objective of this experiment was to determine how well the IDS model, utilizing machine learning algorithms K-Nearest Neighbors (KNN), Random Forest (RF), Extreme Gradient Boosting (XGBoost), and Multi-Layer Perceptron (MLP), could distinguish between the mutually represented classes. The class vector for this experiment is:

$$\left[0\text{: Benign},\quad 1\text{: LDAP},\quad 2\text{: MSSQL},\quad 3\text{: NetBIOS},\quad 4\text{: Syn},\quad 5\text{: UDP},\quad 6\text{: UDPLag}\right]$$

Label	Count
Syn	48840
Benign	46427
UDP	18090
MSSQL	8523
LDAP	1906
NetBIOS	644
UDPLag	55

Label	Count
Benign	51404
UDP	10420
UDPLag	8872
MSSQL	6212
LDAP	1440
NetBIOS	598
Syn	533

Table 6.1: Exp.1 Testing & Training Data distribution

6.2.1 Discussion

The key focus in this experiment was to evaluate the model's performance in distinguishing between the mutually represented classes under controlled conditions. Accuracy, precision, recall, and F1-score were computed for each class and ROC curves were plotted to provide a detailed performance evaluation. Tables 6.2-6.3 and 6.4 sum up all the metrics for this experiment.

Class	Precision	Recall	F1-Score	Support	Precision	Recall	F1-Score	Support
	MLP Classifier				XGBoost			
0	0.9973	0.9983	0.9978	9226	0.9996	1.0000	0.9998	9226
1	0.8741	0.9132	0.8932	380	0.9158	0.9447	0.9301	380
2	0.9182	0.9504	0.9340	1713	0.9451	0.9656	0.9552	1713
3	0.9394	0.9920	0.9650	125	0.9764	0.9920	0.9841	125
4	0.9989	0.9960	0.9975	9861	0.9991	0.9989	0.9990	9861
5	0.9878	0.9724	0.9800	3583	0.9929	0.9785	0.9857	3583
6	0.0000	0.0000	0.0000	9	0.1250	0.1111	0.1176	9
Accuracy		0.9887				0.9929		
M. Avg	0.8165	0.8317	0.8239	24897	0.8506	0.8558	0.8531	24897
W. Avg	0.9886	0.9887	0.9886	24897	0.9930	0.9929	0.9929	24897

Table 6.2: Exp.1: MLP and XGB per class stats

Class	Precision	Recall	F1-Score	Support	Precision	Recall	F1-Score	Support
	Random Forest				KNN			
0	0.9992	0.9999	0.9996	9226	0.9985	0.9995	0.9990	9226
1	0.9577	0.9526	0.9551	380	0.9229	0.9447	0.9337	380
2	0.9399	0.9854	0.9621	1713	0.9343	0.9708	0.9522	1713
3	0.9841	0.9920	0.9880	125	0.9837	0.9680	0.9758	125
4	0.9995	0.9982	0.9988	9861	0.9990	0.9974	0.9982	9861
5	0.9969	0.9766	0.9866	3583	0.9929	0.9752	0.9839	3583
6	0.4286	0.3333	0.3750	9	0.3333	0.2222	0.2667	9
Accuracy		0.9939				0.9919		
M. Avg	0.9008	0.8911	0.8950	24897	0.8807	0.8683	0.8728	24897
W. Avg	0.9940	0.9939	0.9939	24897	0.9920	0.9919	0.9919	24897

Table 6.3: Exp.1: RF and KNN per class stats

All the models showcased excellent statistics in this first approach. Random Forest achieved the highest overall performance, with an accuracy of 99.36% and an F1-score of 0.9937. Also, its ROC-AUC score of 0.9991 indicates excellent ability to distinguish between traffic classes. XGBoost followed closely, with an accuracy of 99.29% and an F1-score of 0.9929, while also achieving the highest ROC-AUC score of 0.9994. KNN and MLP classifiers also performed well, with accuracies of 99.19% and 98.97%, respectively. However, their performance metrics were slightly lower than the other models, and also their run times were

Model	Accuracy	Precision	Recall	F1 Score	ROC AUC	CV Score
Random Forest	0.993654	0.993796	0.993654	0.993672	0.999139	0.993503
KNN	0.991887	0.991991	0.991887	0.991897	0.966639	0.990541
MLP Classifier	0.989677	0.989674	0.989677	0.989607	0.999111	0.988613
XGBoost	0.992891	0.992965	0.992891	0.992909	0.999410	0.991930

Table 6.4: Performance Metrics for Experiment 1

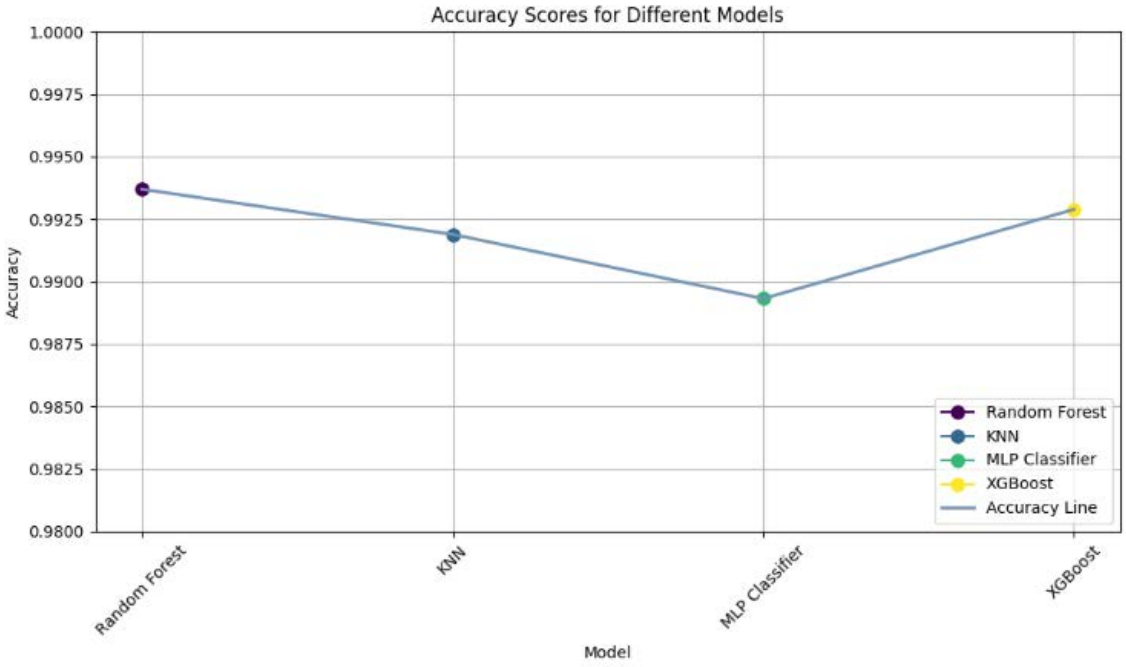

Figure 6.1: Experiment 1: Accuracy of used models

observed to be significantly higher. Notably, the ROC-AUC score for KNN (0.9666) was the lowest compared to the other models, suggesting more difficulty in separating classes.

With this approach and by narrowing the focus to these common traffic types, the models are better positioned to achieve higher precision and recall for challenging classes such as *Syn* flood and *UDP* flood. This approach is particularly important for establishing a baseline performance that reflects the models' effectiveness in handling shared classes across datasets.

However, the exclusion of many of the traffic types reduces the diversity of the dataset, definitely limiting the applicability of the results to real-world scenarios, where network traffic often includes a broader spectrum of patterns. Additionally, the reduced dataset size may affect the robustness of the models, particularly in scenarios requiring generalization to unseen traffic types.

This experiment reduces variability by concentrating solely on common traffic types. It is therefore expected that the model will demonstrate more consistent performance across the mutually represented classes.

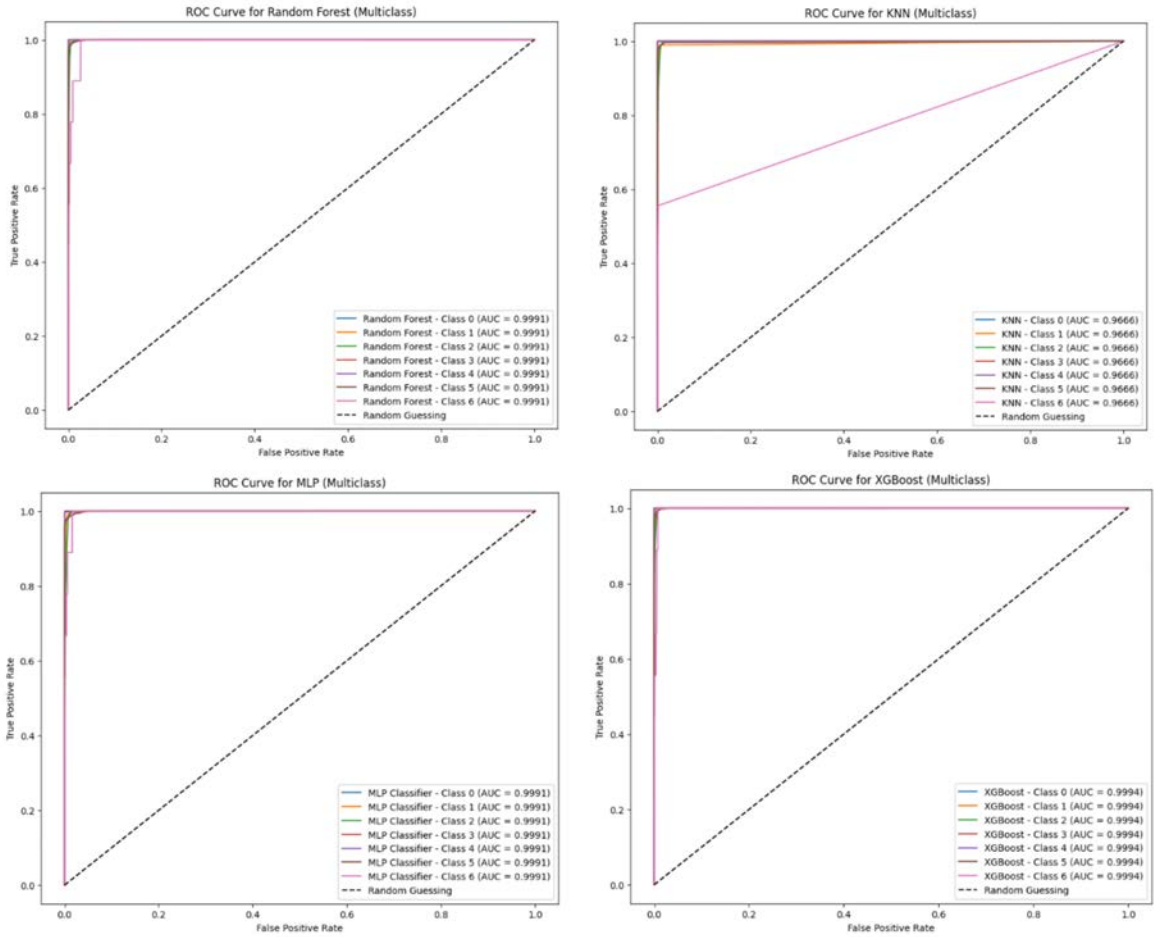

Figure 6.2: Experiment 1: ROC Curves of all models

6.2.2 Review

Becerra et al.[36] utilized the CIC-DDoS2019 dataset similarly to this approach. The authors performed data preprocessing to remove outliers and selected only 22 features using the Pearson correlation coefficient in stage 5.5.2. They also utilized MinMax scaling for data normalization (section 5.5.3). Among 6 models tested including XGB, MLP and RF, the RF demonstrated the best performance with 99.97% accuracy and 99.98% F1 score.

Zinca & Dobrota [38], testing for 6 common classes between the datasets, also exhibited the best average results using RF with 0.9945 precision score for the testing dataset and 0.9979 for the training dataset. A Naïve Bayes model was also tested without much success.

6.3 Exp. 2: Merging the Datasets

In the second experiment, the two distinct datasets were merged to form a unified, more balanced dataset and evaluate the performance of the model in a multiclass classification. This approach involves combining the two separately preprocessed datasets and redistributing the data to ensure more equal representation for all traffic classes. The experiment addresses the challenge posed by severe class imbalance noticed in Experiment 1 (6.2) which can hinder a model's ability to generalize and perform well across all categories.

Label	Count
NTP	121,368
TFTP	98,917
Benign	97,831
Syn	49,373
UDP	28,510
MSSQL	14,735
UDPLag	8,927
DNS	3,669
LDAP	3,346
SNMP	2,717
NetBIOS	1,242
Portmap	685
WebDDoS	51

Table 6.5: Exp.2 Label distribution after label mapping

In this experiment the two cleaned and processed datasets, products of the aforementioned preprocessing procedure, (Table 5.2) were merged into a single super-dataset. In the stage of Label mapping, (5.5.1) pairs of classes being the same under different names were merged (such as *DrDoS_NetBIOS* and *NetBIOS*, *DrDoS_LDAP* and *LDAP*). Table 6.5 illustrates the final distribution of classes before the training phase. The class vector for all 13 unique classes is:

$$\begin{bmatrix} \text{0: Benign,} & \text{1: DNS,} & \text{2: LDAP,} & \text{3: MSSQL,} & \text{4: NTP,} & \text{5: NetBIOS,} & \text{6: Portmap,} \\ \text{7: SNMP,} & \text{8: Syn,} & \text{9: TFTP,} & \text{10: UDP,} & \text{11: UDPLag,} & \text{12: WebDDoS} & \end{bmatrix}$$

6.3.1 Custom splitting, SMOTE & Class weights

In this experiment, a 70%-20%-10% Train-Test-Validation split was implemented, since contrary to Experiment 1, in this case there is only one dataset. This experiment aimed at evaluating and investigating that a ”custom” split of the data at will can be beneficial for training a model with high performance results. It was hypothesized that a structured, customized dataset split could yield higher performance results by allowing finer control over the training data distribution. By balancing the dataset for fairer representation of all classes, this approach aimed to improve the model's generalization ability, particularly for low-frequency attack types that are often overlooked by traditional IDS solutions.

Special measures were taken to address the underrepresented classes, with *Synthetic Minority Over-sampling TEchnique (SMOTE)* [39] being applied to those under a specific threshold. SMOTE is a technique that relies on one of its K-Nearest Neighbors for the minority class to generate synthetic points using this formula:

$$x_{\text{new}} = x_i + \lambda \cdot (x_{nn} - x_i) \tag{6.1}$$

where:

- x_i is an original minority class instance,
- x_{nn} is a randomly selected K-Nearest Neighbor of x_i. The parameter k (typically 5) determines how many neighbors are considered.
- λ is a random value between 0 and 1, ensuring the new sample is positioned along the line segment between x_i and x_{nn}.

For this case, a threshold of *1,000* instances was set, meaning that any class with fewer than 1,000 occurrences was resampled up to 1,000 using SMOTE. This threshold was chosen to strike a balance between preventing the model from completely ignoring minority classes while avoiding excessive over-sampling. By applying SMOTE, three minority classes for the training (70%) split, namely *WebDDoS*, *Portmap*, and *NetBIOS* (which after the splitting was dropped to under *1000* instances) were reinforced, ensuring that the model had a sufficient number of training samples to recognize them.

In conjunction with SMOTE, a *Class Weight* system was applied. This is also a useful technique when dealing with imbalanced datasets, as it helps our model give higher importance to minority classes. Even after applying SMOTE for the 3 classes, there are still significantly fewer samples compared to majority classes. Without class weighting, the model might

still favor the majority classes and struggle to correctly classify the rare attack types. With higher weights for the low classes, the loss function penalizes misclassifications more heavily, and the model becomes more sensitive to those rare categories. This helps improve recall for minority classes, reducing bias toward the dominant classes. Class weights are computed for the training set only, typically using the inverse class frequency formula:

$$w_i = \frac{N}{C \times N_i} \tag{6.2}$$

where:

- w_i is the weight assigned to class i,
- N is the total number of samples in the dataset,
- C is the total number of unique classes,
- N_i is the number of samples belonging to class i.

The following vector presents the calculated class weights for this experiment.

$$\mathbf{w} = \begin{bmatrix} w_{\text{Benign}} \\ w_{\text{DNS}} \\ w_{\text{LDAP}} \\ w_{\text{MSSQL}} \\ w_{\text{NTP}} \\ w_{\text{NetBIOS}} \\ w_{\text{Portmap}} \\ w_{\text{SNMP}} \\ w_{\text{Syn}} \\ w_{\text{TFTP}} \\ w_{\text{UDP}} \\ w_{\text{UDPLag}} \\ w_{\text{WebDDoS}} \end{bmatrix} = \begin{bmatrix} 0.3429 \\ 8.8515 \\ 11.9302 \\ 2.6553 \\ 0.2675 \\ 38.9642 \\ 47.4016 \\ 11.9522 \\ 0.6796 \\ 0.3283 \\ 1.1797 \\ 3.6422 \\ 628.4541 \end{bmatrix}$$

6.3.2 Discussion

The motivation behind this approach was to address the class representation imbalance observed in the first experiment and to potentially mitigate these issues artificially. By combining the datasets, the goal was to create a more analogous representation of traffic patterns across both the training and testing sets. For instance, as shown in Table 5.2, the testing dataset

contains *48,840* instances of *Syn* traffic, whereas the training dataset has only *533*. A similar disparity exists for *UDPLag*, with *55* and *8,872* instances, respectively. In this experiment (Table 6.5), these values were aggregated and then proportionally distributed across the three datasets (*training, testing, and validation*) to ensure a more balanced and realistic representation. Additionally, this experiment sought to evaluate how much a resampling technique like *SMOTE* could positively influence the results and improve model performance.

Class	**Precision**	**Recall**	**F1-Score**	**Support**	**Precision**	**Recall**	**F1-Score**	**Support**
	Random Forest				**KNN**			
0	0.997996	0.998944	0.998470	18941	0.995786	0.998152	0.996968	18941
1	0.656331	0.346049	0.453167	734	0.563380	0.435967	0.491551	734
2	0.488916	0.728440	0.585114	545	0.559464	0.612844	0.584939	545
3	0.844542	0.939493	0.889491	2446	0.862211	0.943990	0.901249	2446
4	0.998184	0.996210	0.997196	24273	0.995928	0.997611	0.996769	24273
5	0.439153	0.500000	0.467606	166	0.492147	0.566265	0.526611	166
6	0.406780	0.350365	0.376471	137	0.527778	0.277372	0.363636	137
7	0.628866	0.674033	0.650667	543	0.703476	0.633517	0.666667	543
8	0.996632	0.990791	0.993703	9556	0.948178	0.974571	0.961193	9556
9	0.999086	0.994490	0.996783	19784	0.998936	0.996563	0.997748	19784
10	0.923663	0.969301	0.945932	5505	0.916113	0.976022	0.945119	5505
11	0.910665	0.737521	0.814998	1783	0.768603	0.475042	0.587175	1783
12	0.062500	0.100000	0.076923	10	0.500000	0.100000	0.166667	10
Accuracy		0.975362				0.969049		
M. Avg	0.719486	0.717357	0.711271	84423	0.756308	0.691378	0.706638	84423
W. Avg	0.976213	0.975362	0.974901	84423	0.967048	0.969049	0.967005	84423

Table 6.6: Exp.2: RF and KNN per class statistics.

In this second experiment, the models achieved high scores but only for the major classes like *Benign, NTP* and *TFTP*. For minority classes like *WebDDoS, NetBIOS* and *Portmap,* performance was suboptimal, with precision and recall below *0.5* in most cases. The models struggled with these underrepresented classes despite SMOTE oversampling. RF and KNN performed consistently well for major classes, showcasing effectiveness for larger class distributions (Table 6.6). The MLP Classifier achieved near-perfect scores for dominant classes and showed slight improvements for some minority classes like *LDAP* and *NetBIOS*. However, it failed to classify the extremely rare classes, showing the insufficiency of SMOTE alone in generating synthetic samples for such classes. XGBoost stood out as the best-performing model overall, achieving the highest accuracy (0.9783), weighted F1 score

Class	Precision	Recall	F1-Score	Support	Precision	Recall	F1-Score	Support
	MLP Classifier				XGBoost			
0	0.992231	0.997941	0.995078	18941	0.998049	0.999314	0.998681	18941
1	0.495192	0.140327	0.218684	734	0.657620	0.429155	0.519373	734
2	0.448366	0.629358	0.523664	545	0.595406	0.618349	0.606661	545
3	0.836553	0.920687	0.876606	2446	0.883454	0.945217	0.913293	2446
4	0.994773	0.995757	0.995265	24273	0.997982	0.998105	0.998043	24273
5	0.443966	0.620482	0.517588	166	0.500000	0.463855	0.481250	166
6	0.866667	0.094891	0.171053	137	0.514286	0.394161	0.446281	137
7	0.519897	0.745856	0.612708	543	0.617211	0.766114	0.683648	543
8	0.998626	0.988803	0.993690	9556	0.996219	0.992675	0.994444	9556
9	0.999594	0.994642	0.997112	19784	0.999341	0.997018	0.998178	19784
10	0.916924	0.972389	0.943842	5505	0.916285	0.982198	0.948097	5505
11	0.937098	0.735278	0.824010	1783	0.924595	0.735838	0.819488	1783
12	0.000000	0.000000	0.000000	10	0.000000	0.000000	0.000000	10
Accuracy		0.972271				0.978359		
M. Avg	0.726914	0.679724	0.666869	84423	0.738496	0.717077	0.723649	84423
W. Avg	0.972556	0.972271	0.970418	84423	0.978031	0.978359	0.977587	84423

Table 6.7: Exp.2: MLP and XGBoost per class statistics.

(0.9783), and ROC-AUC score (0.997). It balanced the performance across majority and minority classes better than other models (Table 6.7). In general, despite SMOTE, rare classes remained difficult to classify. Oversampling such small classes introduces synthetic samples that may not represent actual patterns, leading to noise. Table 6.8 and Figure 6.3 sum up the basic performance metrics for each model.

Model	Accuracy	Precision	Recall	F1 Score	ROC AUC	CV Score
Random Forest	0.975362	0.976213	0.975362	0.974901	0.991893	0.976006
KNN	0.969049	0.967048	0.969049	0.967005	0.951978	0.969017
MLP Classifier	0.972271	0.972556	0.972271	0.970418	0.996158	0.972112
XGBoost	0.978359	0.978031	0.978359	0.977587	0.997882	0.977948

Table 6.8: Performance Metrics for Experiment 2

6.3.3 Review

In [40], the researchers implemented a similar feature selection using PCC (chapter 5.5.2) as well as minority oversampling using SMOTE. Their models' overall accuracy, precision,

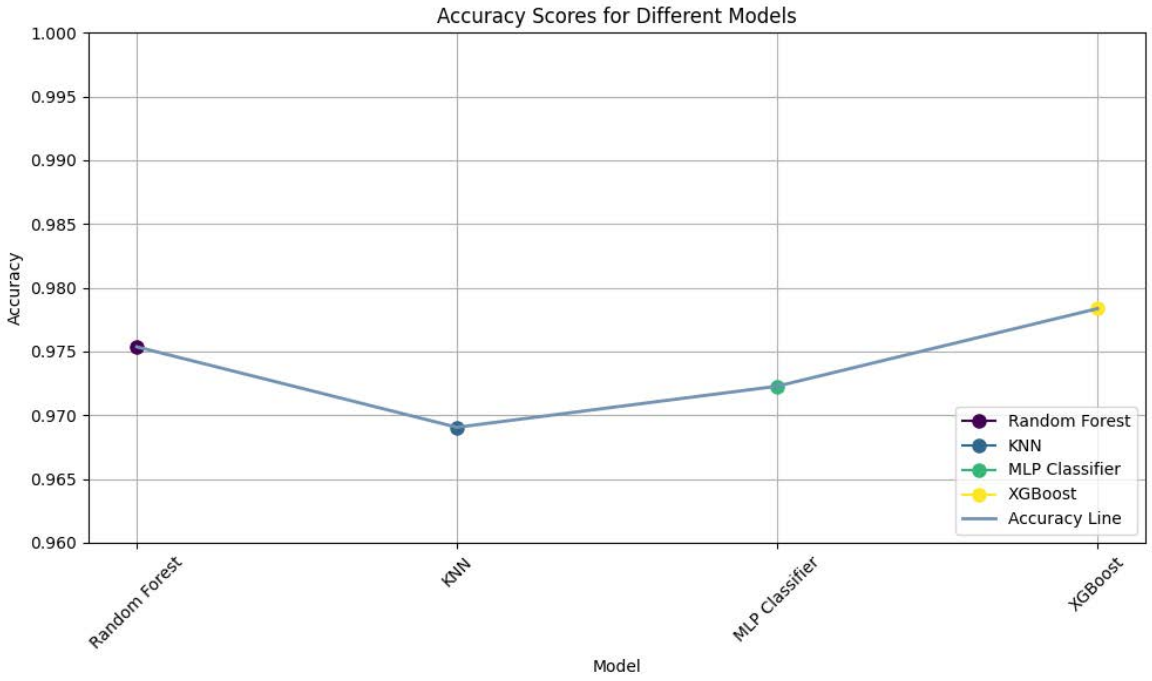

Figure 6.3: Experiment 2: Accuracy of used models

recall and F1-score were highest for RF at 99.99%, 100%, 100%, 100%. KNN also demonstrated good results with 99.93% accuracy, 100% precision, 99% Recall and F1 score.

A hybrid approach, using both SMOTE and TOMEK links [41], ranked and selected features based on the feature shuffling method, using RF to calculate each feature's AUC and selecting it if the value was greater than 0 (AUC>0). This research ranked RF second in all stats, with 99.86% accuracy, 99.86% precision and 99.78% F1 score, only slightly behind Decision Tree (DT). KNN was also employed with 99.82% accuracy.

According to the above, potential improvements for this Experiment 2 could be the pairing of SMOTE with noise-reduction techniques like Tomek Links. It is obvious that altering the SMOTE strategy, i.e. changing the *Threshold* to another number than *1000* or expanding it to more classes could ultimately yield better results. Also, whether customizing the SMOTE procedure to choose a different k other than the default *5* would improve the performance gap for these classes remains an open question. Additionally, more advanced oversampling methods, such as Adaptive Synthetic Sampling (ADASYN), could be explored.

However, resampling techniques do not reflect the natural distribution of traffic types seen in real-world networks. This could still potentially affect the models' applicability. Besides that, synthetic oversampling of minority classes may introduce noise or redundant patterns, affecting the learning process.

6.4 Exp. 3: Binary Classification

The last approach in this thesis treated the problem as binary, with distinction only between malicious or *Non-benign* and *Benign* traffic. All non-benign classes were merged into one super-class (table 6.9). This approach was motivated by the practical and basic needs of an IDS. That is, the first and most important aspect of an IDS is distinguishing malicious traffic from benign traffic. An IDS being able to recognize classes and patterns of traffic is of no use if it cannot determine whether it is malicious or not. Here, the class vector is:

$$\left[0\text{: Benign}, \quad 1\text{: Non-benign}\right]$$

Label	Count
Benign	46427
Non-benign	78743

Label	Count
Benign	51404
Non-benign	254797

Table 6.9: Exp.3 Testing & Training Data distribution

6.4.1 Hyperparameter Tuning

Hyperparameter tuning was also deemed necessary in this experiment, as it allowed proper alignment of the models to the specific characteristics of the dataset, such as high dimensionality (CIC-DDoS2019 dataset containing numerous features), diverse attack patterns (various types of DDoS attacks) and imbalanced class distribution (benign traffic dominating the dataset). The models would not be able to exploit these characteristics without hyperparemeter tuning, resulting in suboptimal performance. This was achieved by utilizing *GridSearchCV*. Part of *scikit-learn* library, *GridSearchCV* performs an exhaustive search over a specified parameter grid, evaluating each combination using cross-validation.

The tuned hyperparameters were then selected to optimize for *F1 score* performance. The rationale is that opting for Accuracy in an imbalanced dataset such as this would be misleading, as it would favor the majority class. Opting for Precision or Recall alone could lead to poor performance on the other. Virtually, relying on F1 score ensures a balanced evaluation of the model's ability to detect attacks (recall) and minimize false positives (precision). According to the above, the final parameters for each model are portrayed in Table 6.10.

Hyperparameter	Random Forest	KNN	MLP Classifier	XGBoost
Bootstrap	False	-	-	-
Max Depth	None	-	-	6
Min Samples Leaf	1	-	-	-
Min Samples Split	5	-	-	-
Number of Estimators	500	-	-	200
Metric	-	Manhattan	-	-
Number of Neighbors	-	5	-	-
Weights	-	Distance	-	-
Hidden Layer Sizes	-	-	(100, 50)	-
Activation Function	-	-	ReLU	-
Solver	-	-	Adam	-
Alpha	-	-	0.0001	-
Max Iterations	-	-	500	-
Learning Rate	-	-	-	0.2
Subsample	-	-	-	1.0
Colsample Bytree	-	-	-	0.8
Gamma	-	-	-	0
Evaluation Metric	-	-	-	Logloss
Random State	42	-	42	42

Table 6.10: Exp.3 Hyperparameter Settings for All Models

6.4.2 Discussion

The results of this experiment on Table 6.13 confirm the high efficiency of the models, particularly KNN and MLP, for binary classification. As shown on plot 6.5, the consistently high ROC-AUC scores (all above 0.99) indicate excellent ability to distinguish between benign and malicious traffic.

However, both Random Forest and XGBoost exhibited weaker performance due to a noticeable drop in precision and recall for class *0:Benign*, as seen clearly from Tables 6.11, 6.12. Random Forest, despite its overall strong accuracy (0.9734), had a significantly lower precision for the benign class (0.8617) compared to KNN (0.9857) and MLP (0.9680), suggesting a tendency to misclassify benign traffic. Similarly, XGBoost showed a strong overall accuracy (0.9764) but struggled with benign class precision (0.8755), reinforcing the impact of class imbalance during training. These models rely heavily on data splits during training and testing and when the class imbalance is high, they seem more skewed towards predicting

the majority class (non-benign) at the expense of the minority class (benign).

This experiment took advantage of the task's binary nature to enhance generalization while reducing computational complexity. The "black or white" classification allowed models to focus solely on malicious traffic detection without added complexity, resulting in improved robustness and faster training times—particularly beneficial when handling large datasets like CIC-DDoS2019.

Class	Precision	Recall	F1-Score	Support	Precision	Recall	F1-Score	Support
	Random Forest				KNN			
0	0.8617	0.9990	0.9253	49849	0.9857	0.9932	0.9894	49849
1	0.9998	0.9684	0.9838	252625	0.9987	0.9972	0.9979	252625
Accuracy		0.9734				0.9965		
M. Avg	0.9307	0.9837	0.9545	302474	0.9922	0.9952	0.9937	302474
W. Avg	0.9770	0.9734	0.9742	302474	0.9965	0.9965	0.9965	302474

Table 6.11: Exp.3: RF and KNN per class stats

Class	Precision	Recall	F1-Score	Support	Precision	Recall	F1-Score	Support
	MLP Classifier				XGBoost			
0	0.9680	0.9978	0.9827	49849	0.8755	0.9989	0.9331	49849
1	0.9996	0.9935	0.9965	252625	0.9998	0.9720	0.9857	252625
Accuracy		0.9942				0.9764		
M. Avg	0.9838	0.9956	0.9896	302474	0.9376	0.9854	0.9594	302474
W. Avg	0.9944	0.9942	0.9942	302474	0.9793	0.9764	0.9770	302474

Table 6.12: Exp.3: MLP and XGBoost per class stats

Model	Accuracy	Precision	Recall	F1 Score	ROC AUC	CV Score
Random Forest	0.973403	0.977031	0.973403	0.974172	0.997208	0.999108
KNN	0.996509	0.996525	0.996509	0.996514	0.997935	0.998035
MLP Classifier	0.994208	0.994367	0.994208	0.994243	0.998280	0.997002
XGBoost	0.976408	0.979295	0.976408	0.977018	0.999103	0.999290

Table 6.13: Performance Metrics for Experiment 3

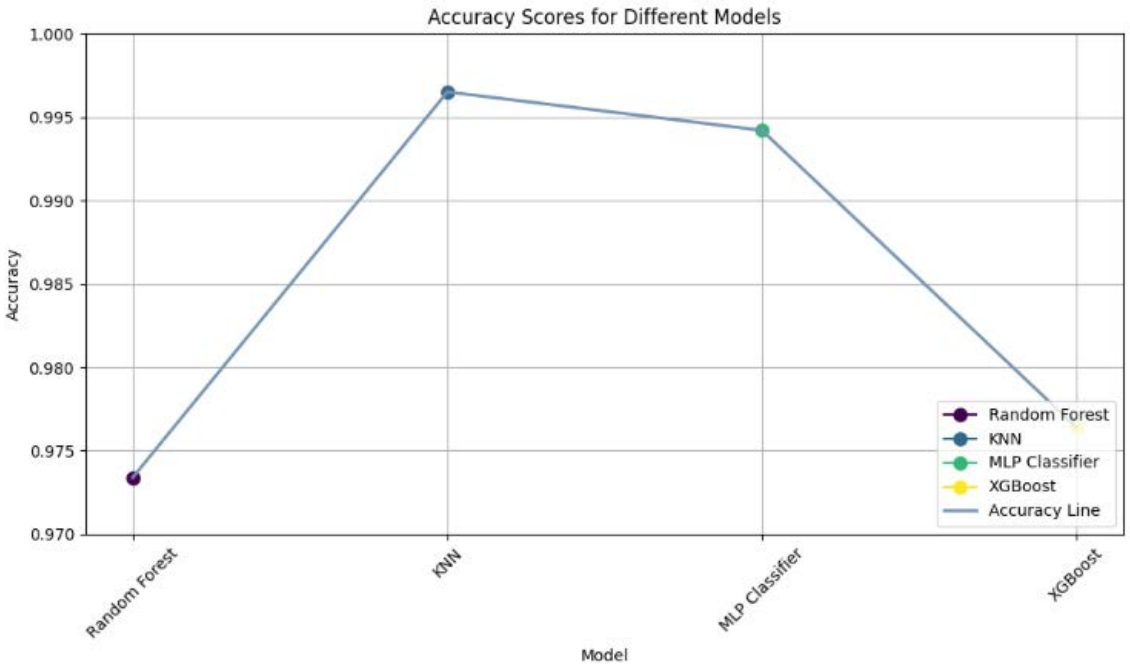

Figure 6.4: Experiment 3: Accuracy of used models

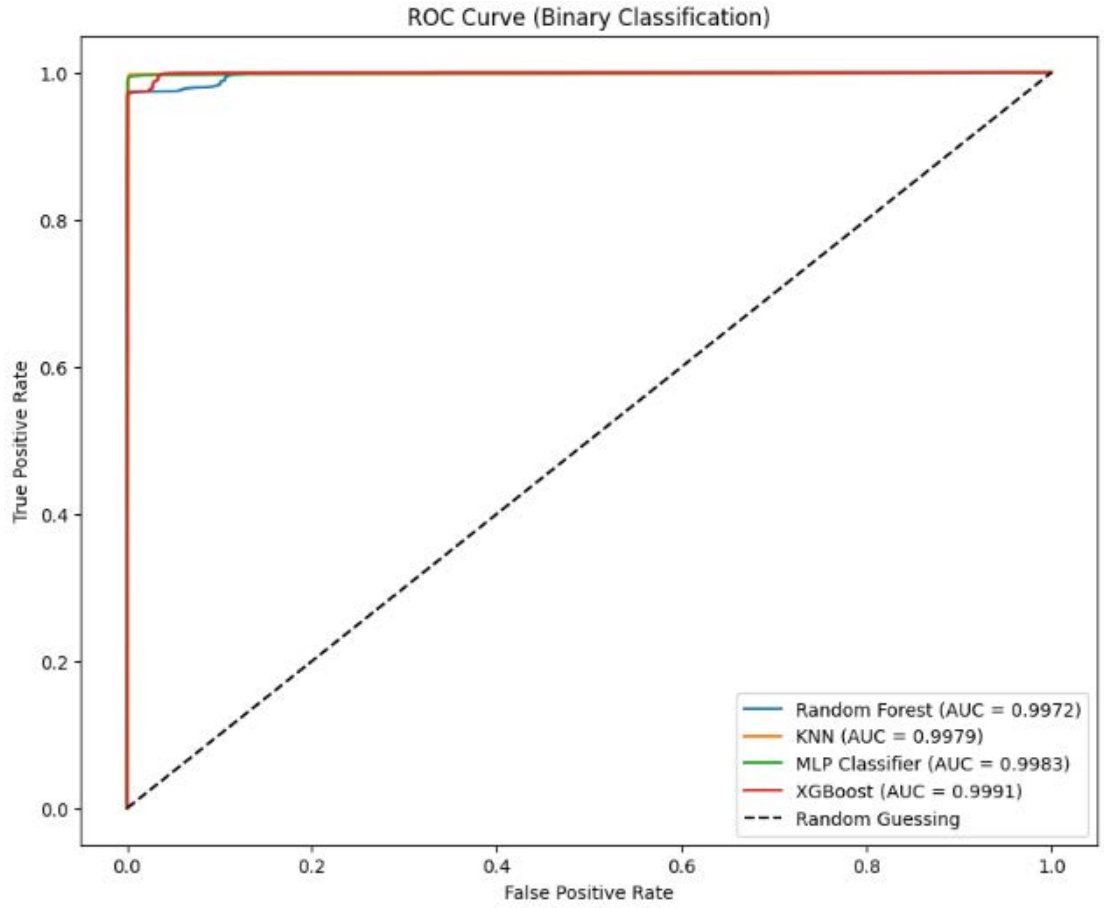

Figure 6.5: Experiment 3: ROC Curves of all models

6.4.3 Review

In [32], Agostinello et al. using the preprocessing strategy of sections 5.3-5.5.2, applied binary classification using DNN, CNN, RNN and exhibited 99.95%, 99.88%, 99.80% accuracy respectively. Their Recall, Precision and Fl scores in each case were on par and almost identical to their respective accuracy scores, very high overall.

Also an experiment for binary classification, [42] achieved nearly perfect scores using RF (0.999992) and J48 (0.999994) algorithms in their research, with XGBoost also showcasing high accuracy (0.99985), recommending them for DDoS detection tasks.

6.5 Conclusions and future work

This thesis explored the application of machine learning techniques in Intrusion Detection Systems (IDS), focusing specifically on the detection of Distributed Denial of Service (DDoS) attacks. These included label mapping, feature scaling, data balancing, and sampling, among others. Four different machine learning models, namely Random Forest, XGBoost, K-Nearest Neighbors, and Multi-Layer Perceptrons, were evaluated across three different experimental setups: training on mutually represented classes across the two datasets, merging the datasets for a unified representation, and binary classification to distinguish between benign and malicious traffic. Each experiment was a completely independent project on its own.

Unlike many studies that prioritize achieving the highest possible performance metrics under almost idealized conditions, this thesis was designed with a real-world application focus. The objective was not to create an artificially optimized model but rather to develop an IDS-ready approach that reflects the practical constraints of deployment. This meant deliberately keeping imperfect yet realistic conditions, including noisy data and class imbalances, instead of applying aggressive data filtering that might lead to unrealistic overfitting scenarios. The methodology ensured that the trained models remain applicable in real-world network environments, where losses are inevitable, and anomaly detection must be tested even under imperfect conditions.

Some of these unorthodox but purposeful decisions that were made throughout this study are explained below. In Experiment 2 (Section 6.3), this study deliberately retained attack classes with minimal representation, even though their presence drove overall metrics downward. The reasoning behind this decision was simple: excluding these rare attack types would artificially boost classification performance, but at the cost of making a future IDS completely ineffective against them. In real-world deployment, partial detection of a rare but critical attack type is far more valuable than complete blindness to it. Another unconventional decision was the application of both SMOTE and class weights—two techniques rarely used together in IDS research. While SMOTE was applied to minority classes to create a more balanced dataset, class weighting was still enforced during training. This approach introduced additional variance, making it harder for the models to achieve inflated accuracy numbers, but ultimately ensured that the model remained sensitive to minority attack types. Many IDS studies opt for only one of these techniques or avoid them entirely to maintain high preci-

sion scores, often at the expense of recall. Furthermore, in Experiment 1 (Section 6.2), where training was conducted on mutually represented classes across datasets, a decision was made to not forcefully equalize class distributions, thereby exposing the models to the natural imbalance present in network traffic. This was a risk to performance metrics, as imbalanced datasets often result in lower recall for minority classes, but it reflected the real-world conditions where some attacks are far more frequent than others. The trade-off was made to ensure the model developed a better understanding of realistic attack distributions, rather than becoming overfitted to an artificially balanced dataset.

Regardless of the above, most of the results were satisfactory and demonstrated that XGBoost consistently achieved slightly higher performance in terms of precision, recall, and AUC metrics. The findings also confirmed the crucial role of feature engineering and data preprocessing in improving model stats. Moreover, the 3 experiments showed that binary classification provided clearer and more reliable detection compared to multi-class classification, even with all classes present.

Despite the promising results, there are several challenges to be resolved. One key, all-time challenge is the deployment of these models in real-time network environments. The models may perform well on preconfigured datasets, but their practical utility lies in handling live traffic. This is where unseen attack patterns may emerge, and their performance remains an open question. Moreover, deep learning approaches could be explored for better performance results. While this thesis primarily focused on traditional machine learning models, deep learning techniques such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformers have shown potential in cybersecurity applications.

6.5.1 Macro vs Weighted average metrics

From the results presented in the experiments, this thesis has purposely measured performance using two distinct and often very divergent metrics. These are the *Macro Average* and the *Weighted Average* of each metric. The inclusion of both of these numbers translates into two different approaches when, ultimately, using one of the trained models as an IDS. It is up to the user to decide what kind of IDS is suitable in each case, based on their unique needs and threat landscape.

The first approach leans towards the *macro average* statistics. This approach focuses on treating all classes equally, regardless of their frequency in the dataset. By calculating the av-

erage performance metric independently for each class and then taking the mean, the macro average approach is particularly advantageous for models designed to excel in detecting **specific types** of attacks or **zero-day** attacks. The macro average approach ensures that even the least common but potentially most dangerous threats are detected and addressed effectively.

Conversely, the second approach focuses on the *weighted average* metrics. This approach gives more importance to the classes based on their sheer frequency. That is, the weighted average approach aligns with the operational reality of an IDS. This approach emphasizes **generalization and versatility**. An IDS designed with a focus on weighted average may not specialize in detecting specific or rare attack types, but it excels in providing consistent and reliable protection across a wide range of more common threats.

6.5.2 Future research: Making a real IDS

It is crucial to mention that training a Machine Learning model differs significantly from setting up a real IDS. The model needs to be applicable to real-world testing, supporting the correct frameworks for this purpose. In this case, exporting each trained model for future use can be achieved using serialization libraries such as *pickle* or *joblib*, which are commonly used in Python. After exporting the models, the *.pkl* files containing the trained models need to be integrated into a system capable of real-time packet analysis.

Here emerges one of the greatest strengths of this thesis, the fact that it is built upon the CIC dataset family, as discussed in section 4.2, a collection of high-quality, real-world cybersecurity datasets. Not only there is huge availability in terms of sheer data available from their research, but also consistency and uniformity. Unlike many other datasets that vary in structure and feature extraction methods, all CIC datasets are captured using the CICFlowMeter (Section 5.2), ensuring that they share a common data format. This means that models trained on one CIC dataset can easily streamline data preprocessing and cleaning and can be easily tested or expanded using another, allowing for seamless adaptation to new threats and scenarios. Moreover, the CICFlowMeter is an open source tool freely available allowing everyone to generate their own attack scenarios by capturing new traffic, and expanding their dataset with custom attack behaviors. This capability is crucial for real-world IDS development. The Packet Capture (*.pcap*) files generated allow the researchers to have access to the full network traffic, giving them plenty of features to select from.

Once the necessary data is on hand, it's time to setup the IDS. Two open source and fully

customizable solutions are *Snort* [43] and *Suricata* [44]. Both of them allow the integration of custom Machine Learning models. Snort, a widely adopted IDS, operates on a single-threaded architecture and utilizes a rule-based detection approach. Suricata, on the other hand, offers native multi-threading capabilities, enabling it to process traffic more efficiently in high-throughput environments. Additionally, Suricata performs deep packet inspection and can leverage most Snort rules with minimal adjustments, providing flexibility in deployment.

Depending on the chosen IDS solution, a lot of implementations and customizations can be performed on this part. A GUI integration with basic HTML/CSS could make the experience more user-friendly. A pre-configured template of custom *CSV* or *PARQUET* files prepared with ready traffic features could allow quicker testing instead of loading a whole dataset everytime. Last but not least, the *alerting rules* that must be set on the IDS. It is, after all, the operator's/administrator's privilege to *overwrite* some of the IDS decisions. For example, no matter if the traffic is malicious or not, the operator might want to be alerted for traffic originating to or from a specific IP in the network. Both solutions offer predefined community rules and the ability to create custom ones.

Ultimately, bridging the gap between theoretical research and practical deployment remains a significant challenge, one that future work can continue to address. Making this transition successfully will not only enhance real-world cybersecurity but also inspire new ideas and innovations for researchers and engineers alike.

Bibliography

[1] TCP Idle Scan (-sI). https://nmap.org/book/idlescan.html, 2025. [Accessed 01-10-2025].

[2] Glen D. Singh, Michael Vinod, and Vijay Anandh. *CCNA Security 210-260 Certification Guide: Build Your Knowledge of Network Security and Pass Your CCNA Security Exam (210-260)*. Packt Publishing, Birmingham, UK, 2018.

[3] Karen Scarfone and Peter Mell. Guide to Intrusion Detection and Prevention Systems (IDPS). NIST Special Publication 800-94, National Institute of Standards and Technology (NIST), 2007.

[4] Robin Sommer and Vern Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy (SP)*, pages 305–316. IEEE, 2010. doi:10.1109/SP.2010.25.

[5] Pavel Laskov and Richard Lippmann. Machine Learning in Adversarial Environments. *Machine Learning*, 81:115–119, 11 2010. doi:10.1007/s10994-010-5207-6.

[6] K Rajasekaran and K Nirmala. Classification and Importance of Intrusion Detection Systems. *International Journal of Computer Science and Information Security*, 10(8):44, 2012.

[7] Hongyu Liu and Bo Lang. Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey. *Applied Sciences*, 9:4396, 10 2019. doi:10.3390/app9204396.

[8] Nguyen Thanh Van, Tran Ngoc Thinh, and Le Thanh Sach. An Anomaly-based Network Intrusion Detection System using Deep learning. In *2017 International*

Conference on System Science and Engineering (ICSSE), pages 210–214, 2017. doi:10.1109/ICSSE.2017.8030867.

[9] Abdulmalik Humayed, Jingqiang Lin, Fengjun Li, and Bo Luo. Cyber-Physical Systems Security—A survey. *IEEE Internet of Things Journal*, 4(6):1802–1831, 2017. doi:10.1109/JIOT.2017.2703172.

[10] Samuel Avwerosughene Arhore. *Intrusion Detection in IoT Systems Using Machine Learning*. PhD thesis, National College of Ireland, 2022.

[11] Thomas H. Ptacek and Timothy N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Technical report, Secure Networks, Inc, 1998.

[12] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of Intrusion Detection Systems: Techniques, Datasets and Challenges. *Cybersecurity*, 2(1):1–22, 2019.

[13] Leslie F. Sikos. Packet Analysis for Network Forensics: A Comprehensive Survey. *Forensic Science International: Digital Investigation*, 32:200892, 2020. doi:10.1016/j.fsidi.2019.200892.

[14] Stefan Axelsson. Intrusion Detection Systems: A Survey and Taxonomy. *Technical Report*, March 2000.

[15] Christopher Krügel and Giovanni Vigna. Anomaly Detection of Web-Based Attacks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pages 251–261. ACM, 2003. doi:10.1145/948109.948144.

[16] Muhammad Ashfaq Khan, Rezaul Karim, and Yangwoo Kim. A Scalable and Hybrid Intrusion Detection System Based on the Convolutional-LSTM Network. *Symmetry*, 11:583, 04 2019. doi:10.3390/sym11040583.

[17] Anna L. Buczak and Erhan Guven. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176, 2015. doi:10.1109/COMST.2015.2494502.

[18] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. A Detailed Analysis of the KDD CUP 99 Data Set. In *2009 IEEE Symposium on Computa-*

tional Intelligence for Security and Defense Applications, pages 1–6. IEEE, 2009. doi:10.1109/CISDA.2009.5356528.

[19] Nour Moustafa and Jill Slay. UNSW-NB15: A Comprehensive Data Set for Network Intrusion Detection Systems (UNSW-NB15 Network Data Set). *2015 Military Communications and Information Systems Conference (MilCIS)*, pages 1–6, 2015. doi:10.1109/MilCIS.2015.7348942.

[20] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking. In *Proceedings of the 6th International Conference on Emerging Networking Experiments and Technologies*, pages 1–12. ACM, 2010. doi:10.1145/1921168.1921179.

[21] S. Kumar, S. Islam, and R. Iqbal. ToN_IoT: The IoT Telemetry Dataset for AI Applications. *arXiv preprint arXiv:2007.14222*, 2020.

[22] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP)*, pages 108–116. SciTePress, 2018. doi:10.5220/0006639801080116.

[23] Mohammadreza MontazeriShatoori, Logan Davidson, Gurdip Kaur, and Arash Habibi Lashkari. Detection of DoH tunnels using Time-series Classification of Encrypted Traffic. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 63–70, 2020. doi:10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00026.

[24] Markus Ring, Sarah Wunderlich, Dominik Grüdl, Dieter Landes, and Andreas Hotho. Creation of flow-based data sets for intrusion detection. *Journal of Information Warfare*, 16(4):41–54, 2017.

[25] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali Ghorbani. Toward Developing a Systematic Approach to Generate Benchmark Datasets for Intrusion Detection". *Computers & Security*, 31:357–374, 05 2012. doi:10.1016/j.cose.2011.12.012.

[26] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Mamun, and Ali Ghorbani. Characterization of Encrypted and VPN Traffic Using Time-Related Features. In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP)*, pages 407–414. SCITEPRESS, 2016. doi:10.5220/0005740704070414.

[27] Sebastián García, Martin Grill, Jan Stiborek, and Alejandro Zunino. An Empirical Comparison of Botnet Detection Methods. *Computers & Security*, 45:100–123, 09 2014. doi:10.1016/j.cose.2014.05.011.

[28] Google Cloud mitigated largest DDoS attack, peaking above 398 million rps | Google Cloud Blog — cloud.google.com. https://cloud.google.com/blog/products/identity-security/google-cloud-mitigated-largest-ddos-attack-peaking-above-398-million-rps, 2023. [Accessed 08-12-2024].

[29] Famous DDoS Attacks: The largest DDoS attacks of all time. https://www.cloudflare.com/learning/ddos/famous-ddos-attacks/, 2024. [Accessed 08-12-2024].

[30] Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, and Ali A. Ghorbani. Developing Realistic Distributed Denial of Service DDoS Attack Dataset and Taxonomy. In *2019 International Carnahan Conference on Security Technology (ICCST)*, pages 1–8, 2019. doi:10.1109/CCST.2019.8888419.

[31] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. Characterization of TOR Traffic Using Time Based Features. In *International Conference on Information Systems Security and Privacy*, volume 2, pages 253–262. SciTePress, 2017. doi:10.5220/0006105602530262.

[32] Davide Agostinello, Angelo Genovese, Vincenzo Piuri, et al. Anomaly-Based Intrusion Detection System for DDoS Attack with Deep Learning Techniques. In *Proceedings of the 20th International Conference on Security and Cryptography*, volume 1, pages 267–275. SCITEPRESS, 2023. doi:10.5220/0012146100003555.

[33] Abdullah Emir Cil, Kazim Yildiz, and Ali Buldu. Detection of DDoS Attacks with Feed Forward Based Deep Neural Network Model. *Expert Systems with Applications*, 169:114520, 2021. doi:10.1016/j.eswa.2020.114520.

[34] Andres Chartuni and José Márquez. Multi-Classifier of DDoS Attacks in Computer Networks Built on Neural Networks. *Applied Sciences*, 11:10609, 11 2021. doi:10.3390/app112210609.

[35] Ebtihal Alghoson and Onytra Abbass. Detecting Distributed Denial of Service Attacks using Machine Learning Models. *International Journal of Advanced Computer Science and Applications*, 12, 01 2021. doi:10.14569/IJACSA.2021.0121277.

[36] Fray Becerra, Ismael Fernández-Roman, and Manuel Forero. Improvement of Distributed Denial of Service Attack Detection through Machine Learning and Data Processing. *Mathematics*, 12:1294, 04 2024. doi:10.3390/math12091294.

[37] Laurens D'Hooge. CIC-DDOS2019-00-Cleaning. `https://www.kaggle.com/code/dhoogla/cic-ddos2019-00-cleaning`, Aug 2022.

[38] Daniel Zinca and Virgil Dobrota. DDoS Attack Detection Using Supervised Machine Learning Alogrithms over the CIDDOS2019 Dataset. *Acta Technica Napocensis*, 63(1):6–11, 2023.

[39] Nitesh Chawla, Kevin Bowyer, Lawrence Hall, and W. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357, 06 2002. doi:10.1613/jair.953.

[40] Sakib Shafin, Sakir Prottoy, Saif Abbas, Safayat Hakim, Abdullahi Chowdhury, and Md Rashid. *Distributed Denial of Service Attack Detection Using Machine Learning and Class Oversampling*, pages 247–259. Springer, 07 2021. doi:10.1007/978-3-030-82269-9_19.

[41] Bidyapati Thiyam and Shouvik Dey. Efficient Feature Evaluation Approach for a Class-Imbalanced Dataset Using Machine Learning. *Procedia Computer Science*, 218:2520–2532, 2023. doi:10.1016/j.procs.2023.01.226.

[42] Mahmood Alfathe, Aida Mustapha, Huthaifa Mohamed, Salama Mostafa, Yousif Yousif, and Ali Shakarchi. Detecting DDoS Attacks in Network Traffic Based on Supervised Machine Learning Techniques. *Al-Noor Journal for Information Technology and Cybersecurity*, 1, 12 2024. doi:10.69513/jnfit.v1.i0.a2.

[43] Martin Roesch. Snort - Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, page 229–238. USENIX Association, 1999.

[44] OISF (Open Information Security Foundation). Suricata - Open Source Network Threat Detection Engine. https://suricata.io. Accessed: 2025-02-03.