MACHINE LEARNING HOST INTRUSION DETECTION:

WEB-BASED THREAT MONITORING

By

Ersad Muçollari

Phd(c). Anxhela Baraj

A THESIS

Submitted to

KOLEGJI UNIVERSITAR "Instituti Kanadez i Teknologjisë"

University College "CANADIAN INSTITUTE OF TECHNOLOGY"

Faculty of Engineering

Department of Software Engineering

In partial fulfillment of the requirements for the degree of:

Bachelor in Computer Engineering and IT

Submitted on September 9th, 2024

Bachelor in Computer Engineering and IT
Submitted on September 9th, 2024

Approved:
Phd(c) Anxhela Baraj, Thesis advisor

Assoc. Prof. Dr. Nihat Adar, Head of Department

I understand that my thesis will become part of the collection of Canadian Institute of Technology. My signature below authorizes release of my thesis to any reader upon request. I also affirm that the work represented in this thesis is my own work.

Ersad Muçollari, Author

Title of thesis: Machine Learning Host Intrusion Detection: Web-Based Threat Monitoring

Abstract:

In today's connected society, maintaining computer networks secure against cyber threats and unauthorized access is one of the most serious issues. The goal of this research is to design and implement a web-based utility for traffic monitoring within network data based on machine learning. The study employs established machine learning models, including the Kitsune autoencoder-based framework and the Random Forest classifier, to detect anomalies in network traffic. The Random Forest model is trained on the CICIDS2017 dataset from the Canadian Institute for Cybersecurity, while Kitsune operates unsupervised, without the need for explicitly labeled data. The process involves data preprocessing, feature engineering, and the integration of these models into a web application for ease of use and visualization. Implementation uses Python and popular machine learning libraries. A web-based front end is created using the Streamlit framework, which takes network traffic files as an input from the user and finds any sort of anomaly in them. Initial outcomes demonstrate that the method can efficiently classify different anomalies in network traffic, implying application potential. Overall, this study shows that integrating machine learning models into user-friendly web-based applications can tremendously improve network security issues by allowing quick and efficient anomaly detection.

Keywords: Host Intrusion Detection; Kitsune; Random Forest; Web-based anomaly detection; Cybersecurity; Web-based application

Acknowledgment

I am deeply grateful to those who supported my thesis journey. Anxhela Baraj's critical feedback on my technical analysis and framework was invaluable. Her dedication and timely responses during busy schedules were instrumental in refining my work, giving me confidence in its significance and quality.

I also appreciate my colleagues and friends whose advice and support were crucial in shaping my research. Their contributions helped me navigate the project's complexities and stay focused.

Lastly, I thank my family. Both of my parents taught me clear and logical thinking, and my sister showed the importance of determination. This work is dedicated to them with sincere gratitude.

Table of Contents

List of Figures

List of Tables

I. Introduction

In today's connected world, network security is crucial in the digital era. As modern infrastructure has turned digital, the threat landscape within it adapts to new, advanced threats as they manifest. Network intrusions leading to events like data breaches, operational outages and financial loss are occurring at an unprecedented pace. Therefore, the aim of this thesis is to tackle these problems by providing novel approaches that can improve network security, using machine learning. The research aims to create a web utility that can categorize malignant network traffic from the web, making it an intuitive and lightweight option for identifying and countering cyber security threats.

I.1 Background

One of the biggest problems, companies face today is network security due to the rapid growth of digital footprints. As the sophistication and volume of cyber threats rise, so do the consequences of network attacks. High-profile incidents like the 2017 Equifax breach, which exposed personal details of 147 million people, show the potentially devastating effects of such breaches. These attacks undermine customer trust and harm organizations' reputations in addition to causing financial losses. Malicious activities encompass phishing scams, malware, and Distributed Denial of Service (DDoS) assaults, each of which presents distinct threats, including but not limited to financial fraud, data theft, and the interruption of vital services (Coulibaly, 2020).

The necessity of threat detection and countermeasures has grown increasingly important as attacks become more frequent and complex. Network security has historically depended on signature-based detection techniques. However, as sophisticated and novel forms of cyberattacks proliferate, these systems are becoming less and less useful. Networks are exposed to new threats since signature-based solutions are limited in their ability to identify known threats (Kumar et al., 2021). This drawback highlights the need for more sophisticated, adaptable security solutions that can detect and address new threats in actual network traffic (Ullah et al., 2021).

I.1.1 Introduction to Intrusion Detection and Prevention

In order to detect potential security incidents, such as violations or impending threats to computer security regulations, acceptable usage standards, or standard security practices, intrusion detection involves tracking events that occur within a computer system or network and analysing them. These occurrences can originate from a number of different sources, including as malware such as viruses and spyware, attackers utilising the Internet to achieve unauthorised access, or even authorised individuals abusing their privileges or trying to obtain more unauthorised powers. While a large number of events are malicious, some can be unintended, like when someone types the erroneous computer address and tries to connect to the incorrect system (Abbas et al., 2023).

An Intrusion Detection System (IDS) is software that automates this monitoring and analysis process. An Intrusion Prevention System (IPS) goes a step further by not only detecting potential incidents but also attempting to prevent them. This section provides a comprehensive overview of IDS and IPS technologies, starting with their applications, followed by a discussion of their key functions and detection methodologies. It also outlines the major classes of IDS and IPS technologies.

Both IDS and IPS technologies share many capabilities, and administrators can often disable the prevention features in IPS products to make them function as IDSs. Therefore, for simplicity, the term Intrusion Detection and Prevention Systems (IDPS) is used to refer to both IDS and IPS technologies throughout this guide, except where specific distinctions are necessary (Abbas et al., 2023).

I.1.2 Understanding Anomaly-Based Detection

Anomaly-based detection is a method where definitions of normal activity are compared with observed events to identify significant deviations. An Intrusion Detection and Prevention System (IDPS) employing this method generates profiles representing normal behavior for various entities such as users, hosts, network connections, or applications. These profiles are developed by monitoring typical activities over a certain period. For example, a network profile might reveal that web activity generally consumes 13% of network bandwidth during typical workday hours. The IDPS then utilizes statistical methods to compare current activities against these established profiles,

detecting anomalies when, for example, web activity consumes significantly more bandwidth than expected and subsequently alerting an administrator (Al Jallad et al., 2020).

The capacity of anomaly-based detection to recognise hitherto unidentified dangers is a key benefit. A computer infected with a new kind of malware, for instance, might use up a lot of processing power, send a lot of emails, and create a lot of network connections—all actions that would significantly differ from the computer's pre-established profile. Over the course of the training phase, which usually lasts a few days or weeks, an initial profile is established. These profiles might be dynamic, meaning they constantly alter in response to newly detected events, or static, meaning they stay the same unless manually updated. Static profiles require regular updates as systems and networks change; dynamic profiles avoid this problem but leave you open to attackers who might progressively increase malicious activity and make it appear normal to the system (Al Jallad et al., 2020).

However, the unintentional inclusion of malicious actions in the profiles during the training phase is a prevalent issue with anomaly-based IDPS. Furthermore, because computing tasks are complicated, it can be difficult to adequately capture regular behaviours. For instance, the system may incorrectly mark an infrequent maintenance task that involves big file transfers as a substantial deviation when it does occur since it may not have been recorded during the training phase. As such, anomaly-based IDPS frequently produces a large number of false positives, particularly in contexts that are dynamic and diverse. The difficulty analysts encounter in identifying the cause of an alarm and verifying its veracity as a result of the intricacy and quantity of events that could trigger an alert is another significant problem (Al Jallad et al., 2020).

I.2 Problem Statement

The weaknesses of the available network intrusion detection techniques in thwarting sophisticated and dynamic cyberthreats are discussed in this thesis. Due to the increased sophistication of these threats, signature-based detection systems are frequently overwhelmed by new attack types, which increases the frequency of false positives and false negatives. The shortcomings of these antiquated systems underscore the necessity for more sophisticated detection techniques that go beyond basic signatures in order to successfully manage the dynamic cyber threat environment of today. Using machine learning to create systems that can learn from large amounts of data is a promising option.

Through the analysis of network traffic patterns and activities, machine learning models are able to identify known and undiscovered potential dangers that are often overlooked by traditional methods. This research focuses on creating a web-based tool that employs machine learning for network intrusion detection, enhancing organizational security.

I.3 Objectives

To design and implement a web-based utility for classification of abnormal network traffic, employing machine learning models to secure network against intruders, might include:

- Study the Feasibility of Machine Learning Models: Analyze the potential of machine learning models, such as Random Forest and Kitsune, for detecting anomalies in network traffic. This analysis will compare accuracy and practicality.
- Improve Usability and Accessibility: Ensure the tool is easy to use for individuals without diverse technical skills by creating user-friendly interfaces and effective visualizations, by identify different threats.
- Increase Detection: Demonstrate that the Kitsune Autoencoder-Based Framework implementation can detect more threats, and do so with greater accuracy and lower overhead than traditional methods. This should be accompanied by a comprehensive testing and validation (with real network traffic) process.

The core objectives of the research are to make the tool more useful, easy-to-use for network intrusion detection which would impact the cybersecurity, enabling organization to secure their digital assets.

I.4 Motivation

The motivation to do this research comes from the demand for more advanced network security solutions to keep up with the changing cyber threat scenario. However, discovery and remediation of intrusions is the holy grail for network administrators and security professionals alike; This is because of the fact that traditional methods are not enough anymore need for a more sophisticated tool to develop their processes to be able to eliminate defects accurately. This is the gap we seek to fill with this research: a web-based utility that can improve the intrusion detection through machine learning. The network administrator community is one that typically experiences challenges due to

the nature of their work and the expected results of this research effort offer a method that is both effective and easy to use. Being web-based, the utility can be opened on any device with an internet connection, which offers flexibility and convenience. Along with the possibilities the research has for enhancing cybersecurity practices through new forms of anomaly detection in network traffic. Combining an emphasis on user-friendly design with sophisticated machine learning integration, the research stands to have meaningful implications for network security as a whole and help organizations build better defenses against cyber security threats.

II. Literature Review

II.1 Understanding Host Intrusion Detection

One crucial component of cybersecurity is host intrusion detection, which looks for illegal activity within a network. Three main strategies have been used by intrusion detection systems (IDS) in the past: hybrid methods, anomaly-based detection, and signature-based detection.

Predefined patterns or known threat signatures are used in signature-based detection. Signature-based systems are good at spotting known assaults, like those found by tools like Snort and Suricata, but they have trouble identifying new or unidentified threats. For example, Snort is commonly used to match network traffic against known signatures in order to detect malware traffic. Recent research, however, shows that although Snort is still successful against known threats, it has difficulties against sophisticated attack patterns and high-speed networks (Al Waili, 2023) (Erlacher & Dressler, 2022). The main drawback of signature-based systems is that they frequently produce a large percentage of false negatives due to their incapacity to identify novel or emerging threats like polymorphic or zero-day malware.

By enhancing detection accuracy and correcting the shortcomings of conventional systems, recent research has suggested machine learning strategies to get over these restrictions(Akhtar & Feng, 2022).

Anomaly-based detection establishes a baseline of normal network behavior and identifies deviations from this norm as potential threats. This approach can detect unknown threats but often results in high false positive rates due to the variability of normal network behavior. Techniques such as statistical modeling, clustering, and machine learning are commonly used in anomaly-based systems (Liao, Lin, Lin, & Tung, 2013).

To increase detection accuracy and lower false positives, hybrid systems combine the best features of anomaly- and signature-based techniques. These systems offer a more complete security solution by combining behavioural analysis and predetermined signatures. The efficacy of hybrid systems has been shown by recent studies, especially when signature-based detection and unsupervised anomaly detection are combined. This lowers false alarm rates and increases detection rates for both known and undiscovered threats (Bhadauria & Mohanty, 2021). Although hybrid approaches perform better, they are frequently more intricate and computationally demanding.

Machine-learning techniques can now be used to significantly enhance detection capabilities thanks to recent developments in network intrusion detection. Without depending exclusively on predetermined signatures, machine learning models—especially unsupervised techniques—show promise in identifying unknown assaults.

Machine learning models, especially unsupervised methods, show promise in detecting unknown attacks without relying solely on predefined signatures. These models learn and adapt as new network behavior patterns emerge, significantly enhancing detection in dynamic environments (Rele & Patil, 2023).

II.2 Machine Learning in Cybersecurity

Machine learning has proven to be a very effective tool in improving cybersecurity mechanisms. Utilizing data that they can learn from, machine learning models can detect and stop cyber threats better than traditional means. Unsupervised techniques, such as clustering techniques (K-means, hierarchical clustering) and anomaly detection techniques, are particularly relevant for discovering various patterns in network traffic without any labeling (Chandola, Banerjee, & Kumar, 2009).

One approach to automation that has been shown to be effective for implementing machine learning in cybersecurity is the use of clustering models to group similarly related network events to indicate an attack in progress. Also, the use of Support Vector Machines (SVM) for the classification of network traffic as normal and abnormal has shown high detection accuracy of threats.

However, even with the promising benefits, integrating machine learning into cybersecurity faces its own challenges. A major obstacle is the requirement for massive quality data to sufficiently train the models. Further, machine learning models are susceptible to adversarial attacks—situations in which input data are manipulated by an adversarial entity to thwart the system. In order to tackle these challenges, we need continuous research and development of machine learning models which can handle such unfavorable circumstances in an efficient way.

II.3 Anomaly Detection Approaches

Anomaly detection is essential in host intrusion detection, with numerous approaches proposed to enhance its performance. This section considers Random Forest and autoencoders for their comprehensive approaches and capabilities.

Traditionally, intrusion detection approaches are studied from two main perspectives: anomaly detection and misuse detection, with minimal distinction between their characteristics. (Liao, Lin, Lin, & Tung, 2013) further categorized these approaches into three subcategories: computation-dependent, artificial intelligence, and biological concepts. However, this classification does not comprehensively capture the properties of detection approaches. To address this, we propose a more detailed classification consisting of five subclasses, each offering a deeper perspective on their characteristics: statistics-based, pattern-based, rule-based, state-based, and heuristic-based approaches. Table 2 organizes current intrusion detection methods based on this viewpoint.

In Table 2, the time series field indicates whether the approach considers time series behavior. The detection of attacks field specifies the types of attacks identifiable by each approach. The performance field evaluates the efficiency of IDS in processing audit events. The type of source includes audit data, user profiles, security policies, and knowledge from previous attacks, which can help distinguish intrusion behaviors from suspicious activities. Additional specialized characteristics for each technique are listed in other characteristics.

Recent works have integrated several detection approaches from these five subclasses into more sophisticated methods to enhance efficiency and reduce false alarm rates compared to individual approaches.

An ensemble learning technique called Random Forest is employed to identify outliers. By focussing on both normal and abnormal data points, it learns patterns from the complete dataset to identify anomalies. This method works well for online intrusion detection since it is efficient, requires less calculations, and summarises big datasets well. Autoencoders are used by Kitsune, a lightweight online host intrusion detection system, to detect anomalies. Neural networks called autoencoders compress and rebuild data, then use the reconstruction mistakes to find anomalies. Kitsune improves detection accuracy and lowers false positives by fusing the advantages of autoencoders with an ensemble learning strategy.

It is perfect for environments with fluctuations in the network due to its dynamic adaptability.

Because autoencoders may train unsupervisedly, they are frequently utilised independently in anomaly detection. When dealing with complex, high-dimensional data, where standard techniques might perform poorly, they are especially effective. By separating normal data from abnormalities, autoencoders' reconstruction error measure helps build a reliable pipeline for intrusion detection.

In contrast, autoencoders, Random Forest, and Kitsune each have special benefits. Autoencoders are excellent at organising complex data patterns, Random Forest is highly scalable and computationally economical, Kitsune offers lightweight adaptive high detection accuracy, and all of these technologies have a broad range of applications in intrusion detection scenarios.

The reasons behind the selection of autoencoders, Random Forest, and Kitsune in this study are their complementing features. Autoencoders find complex anomalies, Kitsune delivers flexibility and visualisation, and Random Forest offers speed and scalability. When combined, these methods create a thorough model for host intrusion detection that improves security overall and tackles a variety of issues. Table 1 provides a complete overview of the advantages and disadvantages of different intrusion detection technologies. Table 2 presents additional comparisons between various types of IDS technology.

Table 1.Pros and Cons of Intrusion Detection Methodologies. (Liao, Lin, Lin, & Tung, 2013)

Signature-based (knowledge-based)	**Anomaly-based (behavior-based)**	**Stateful protocol analysis (specification-based)**
Pros	**Pros**	**Pros**
- Simplest and effective method to detect known attacks. - Detail contextual analysis.	- Effective to detect new and unforeseen vulnerabilities. - Less dependent on OS. - Facilitate detections of privilege abuse.	- Know and trace the protocol states. - Distinguish unexpected sequences of commands.
Cons	**Cons**	**Cons**
- Ineffective to detect unknown attacks, evasion attacks, and variants of known attacks. - Little understanding of states and protocols.	- Weak profiles accuracy due to observed events being constantly changed. - Unavailable during rebuilding of behavior profiles.	- Resource consuming to protocol state tracing and examination. - Unable to inspect attacks looking like benign protocol behaviors.

- Hard to keep signatures/patterns up to date.	- Difficult to trigger alerts in right time.	- Might be incompatible with dedicated OSs or APs.

The advantages and disadvantages of three intrusion detection techniques—signature-based, anomaly-based, and stateful protocol analysis—are compiled in Table 1. While signature-based detection works well against known assaults, it is not as reliable against unexpected ones and needs to be updated often. While anomaly-based detection is capable of finding novel vulnerabilities, it has issues with timely alarms and profile correctness. Although stateful protocol analysis is resource-intensive and might not work with some systems, it provides comprehensive protocol state tracing (Liao, Lin, Lin, & Tung, 2013).

Table 2.Comparisons of IDS Technology Types. (Liao, Lin, Lin, & Tung, 2013)

Item	**Technology**			
	HIDS	**NIDS**	**WIDS**	**NBA**
Components	Agent: software (inline) MS: 1 n DS: 1 n (option)	Sensor: n (inline/passive) MS: 1 n DS: 1 n (option)	Sensor: n (passive) MS: 1 n DS: 1 n (option)	Sensor: n (most passive) MS: 1 n (option) DS: optional
Detection scope of sensor/agent	Single host	Network subnet:n Host: n	WLAN:n WLAN client: n	Network subnet: n
Architecture	MN or SN	MN	MN or SN	MN or SN
Strengths	Only HIDS can analyze end-to-end encrypted communications' activity.	Capable to analyze the broadest scopes of AP protocols	WIDS is more accurate due to its narrow focus. Only WIDS can supervise	Superior detection powers at reconnaissance scanning, reconstruct

			wireless protocol activity.	malware infections and DoS attacks
Technology limitations	More challenging in detection accuracy due to a lack of context knowledge Delays in alert generation and centralized reporting Consume host resources Conflict with existing security controls	Cannot monitor wireless protocols High false positive and false negative rates cannot detect attacks within encrypted traffic	No full analysis support under high loads. Cannot monitor AL, TL and NL protocol activities. Cannot avoid evasion techniques. Sensors are susceptible to physical jamming attacks. Cannot compensate for insecure wireless protocols	The major limitation is the delay in detection attacks, caused by transferring flow data to NBA in batches, but not in real time.
Security capabilities and Information gathering	Network traffic, system calls, file system activity.	Hosts, OSs, APs, network traffic	WLAN, devices (e.g., APs, clients)	Hosts, OS, services (IP, TCP, UDP, etc).

Logging	Reference (Stavroulakis and Stamp, 2010)	Reference (Stavroulakis and Stamp, 2010)	Reference (Stavroulakis and Stamp, 2010)	Reference (Stavroulakis and Stamp, 2010)
Detection methodology	SD and AD (combined)	SD (major), AD and SPA	AD (major), SD and SPA	AD (major), SPA
Type of suspicious events detected	AL, TL and NL network traffic, event logs (e.g., application activities, file system activities), system logs (e.g., configurations, OS activity)	AL, TL, NL and HW reconnaissance and attacks, unexpected AP services, policy violations	Wireless protocol activity, insecure WLAN and devices, DoS attacks, network scanning, policy violations	AL, TL, NL anomalous traffic flows (DoS attacks, malware), unexpected AP services, network scanning, policy violations

Table 2 presents a comparison of various Intrusion Detection System (IDS) technologies. Network Behaviour Analysis (NBA), Wireless IDS (WIDS), Host-based IDS (HIDS), and Network-based IDS (NIDS). Every technology is assessed on a number of factors, such as its architecture, components, detection range, strengths, weaknesses, and security prowess. Although it can examine end-to-end encrypted communications on a single host, HIDS has limitations with regard to resource usage and detection accuracy. NIDS is capable of monitoring hosts and network subnets for a wide range of application protocols, but it is slow in high-traffic situations and is unable to decipher encrypted communications. WIDS is very good at accurately tracking wireless protocol activity, but it cannot make up for unsecure wireless protocols and is susceptible to physical jamming. NBA is highly effective at detecting reconnaissance scanning and reconstructing malware infections, yet it suffers from delays in detection due to the batch processing of flow data (Liao, Lin, Lin, & Tung, 2013).

II.4 Challenges of Using Machine Learning

Considering their substantial academic study, anomaly detection systems' poor effectiveness in operational situations is surprising. In other fields, including commercial settings, where they manage massive volumes of data that make manual inspection unfeasible, machine learning methods have proven to be quite effective. The disparity results from the special difficulties that are inherent in the intrusion detection field. While machine learning algorithms are good at finding similarities, they have trouble spotting outliers. Collaborative filtering, for example, is used by product recommendation systems such as Amazon's to recommend products based on previous purchases. On the other hand, anomaly detection necessitates determining goods that are not commonly bought together, which makes it less precise and easier to use. Furthermore, training on samples from all classes is usually necessary for machine learning. The necessity of a comprehensive and accurate model of normality further complicates this, as any deviation can result in misclassification, thus compromising the system's reliability (Sommer & Paxson, 2010).

Errors in intrusion detection systems have serious consequences. While false negatives might cause serious harm, false positives need expensive analyst effort to analyse trivial instances. For instance, inaccurate recommendations are generally safe in product recommendation systems. However, as demonstrated by Amazon's recommendation engine, which accepts a high error rate, even a minor false positive rate in intrusion detection might make a system unworkable. The significance of precise and reliable anomaly detection is highlighted by the high cost of errors in intrusion detection systems. Semantic gaps are also caused by anomaly detection systems' inability to generate information that network operators can use. Frequently, these algorithms detect departures from typical conduct without evaluating their importance. Effective intrusion detection should bridge this gap to meet operational needs by incorporating local security policies, which vary widely across different environments. This necessity of understanding the network's semantics is crucial yet often overlooked in academic research. The challenge of translating technical detections into actionable intelligence for network operators is a significant barrier to the successful deployment of machine learning in this field (Sommer & Paxson, 2010).

The intrinsic diversity of network traffic makes anomaly identification more difficult. The variability of traffic variables, such as bandwidth, connection time, and application mix, makes it difficult to develop a reliable "normal" model. While longer durations of data aggregation can help reduce this

fluctuation, this method frequently yields noisy results. This unpredictability emphasises how crucial it is to look into more straightforward, non-machine learning strategies that could be just as successful. Anomaly detection system evaluation is made more difficult by the dearth of relevant publicly available datasets. The few datasets that are still accessible don't represent current network activity; one example is the out-of-date packet traces from DARPA/Lincoln Labs. It can be difficult for researchers to develop their own datasets when they don't have access to big networks. Data sharing and evaluation are further complicated by the sensitive nature of network data. The combination of these factors makes it difficult to conduct thorough and representative evaluations of anomaly detection systems, thereby hindering their development and deployment (Sommer & Paxson, 2010).

Attackers constantly adapt to evade detection systems in the adversarial environment in which intrusion detection operates. Evasion is a serious theoretical issue, but as most attacks are not highly focused, their actual impact may be less severe. However, resolving this adversarial component is essential to enhancing machine learning's efficacy in intrusion detection. Attackers and defenders are engaged in a never-ending arms race because attackers can modify their tactics to avoid discovery. Because of this dynamic, intrusion detection systems need to be able to predict and adjust to new threats in addition to identifying recognised ones. The intricacy and dynamic character of these hazards present an additional challenge to the efficient implementation of machine learning within this field. (Sommer & Paxson, 2010).

II.5 Multi-dimensions threats classification model

The majority of security threat classifications usually only use one or two criteria, which leads to a list that is not comprehensive and does not include all threats or categories that are mutually incompatible. This is insufficient for dynamic contexts where insider threats are a serious worry, even though it might work in somewhat stable environments with little organisational change. It is imperative to identify all threat characteristics in order to properly reduce risks that organisations face, as they might have a detrimental influence on their operations and reputation. By enabling proactive security, proper classification enables organisations to comprehend the dangers that impact their assets and the places they could effect. Additionally, it helps managers create information systems that are less vulnerable. However, existing classifications often fail to adhere to

classification principles, necessitating the creation of hybrid models that combine different classification schemes (Jouini, Ben Arfa Rabai, & Ben Aissa, 2014).

To address these challenges, Jouini, Ben Arfa Rabai, and Ben Aissa (2014) proposed a hybrid model for information system security threat classification called the Multi-dimensions Model for Threat Classification. This model respects all threat classification principles by combining various criteria and highlighting their potential impacts. The criteria include the source of the threat (internal or external), the agents causing the threats (human, environmental, technological), the motivation behind the threats (malicious or non-malicious), the intention (intentional or accidental), and the impacts (destruction, corruption, theft/loss, disclosure, denial of use, elevation of privilege, illegal usage).

Threats are first categorised based on their source, which can be either internal or external, using these criteria. For example, environmental risks may arise from natural processes occurring within the organisation or from natural processes occurring outside the system boundaries. These dangers are normal and usually not intended to cause harm. On the other hand, human-made activities are further divided depending on intent: intentional or unintentional, and are further defined by the user's objective, whether malicious or non-malicious. Technological hazards originate from materials-affecting physical and chemical processes, which are typically unintentional and lack malicious intent. The final criterion in the model addresses the impact of threats, which can be categorized into seven distinct types: destruction of information, corruption of data, theft or loss of information, unauthorized disclosure, denial of access, privilege escalation, and illegal usage.

This thorough categorisation model aids in the identification and mitigation of the various security threats that organisations encounter. Through comprehension of the source, characteristics, and consequences of every danger, institutions can formulate more resilient plans to safeguard their resources and uphold their operational integrity. For example, determining the nature of an external or internal threat aids in developing customised defences, and recognising the purpose of threats helps direct preventative actions. Understanding the motivations behind human behavior—intentional or not—helps develop more focused and efficient response strategies. Because repercussions are included in the model, organisations are guaranteed to be able to foresee and be ready for any number of ways that risks may materialise, from unauthorised access to data corruption.

III. Methodology

III.1 Overall Research Strategy

Two machine learning models are developed in the process of developing a web-based host intrusion detection (HID) application in order to offer a flexible and reliable solution that can be used in a variety of settings. The Random Forest classifier is the first model selected, and it will be trained using the CICIDS2017 dataset provided by the Canadian Institute for Cybersecurity (CIC). This dataset ensures a representative simulation of real-world scenarios by providing a complete collection of real-world data that includes prevalent current threats and realistic background traffic.
To find anomalies in network traffic, an autoencoder-based system based on unsupervised learning methods will be put into place concurrently. This approach excels in identifying trends in data and highlighting deviations that can point to possible security issues.
Streamlit would be used to construct a web application that would include these models into an approachable platform. Network traffic monitoring and interactive visualisation would be possible with this platform. The Random Forest model will be created and trained after the dataset has been cleaned and preprocessed to make it ready for analysis. In parallel, the trained models would be put into the application and the Streamlit environment would be configured. To facilitate user interaction with the models and facilitate effective analysis of network traffic data, an intuitive user interface would be built.
The comparative examination of these models would concentrate on assessing their applicability and effectiveness in cybersecurity scenarios. Because the Random Forest classifier can handle huge datasets and feature selection well, it is likely to perform robustly in recognising known forms of attacks. On the other hand, the autoencoder model would probably be better at finding new or undiscovered anomalies, which would enhance the capabilities of the Random Forest model. By taking a holistic approach, the system's overall security posture will be improved by ensuring extensive network traffic analysis.

III.2 Dataset Utilization

We developed and tested our Random Forest intrusion detection classifier using the CICIDS2017 dataset from the Canadian Institute for Cybersecurity. Because it offers trustworthy test data that replicates actual network traffic, including both routine activity and frequent cyberattacks, this dataset is crucial. Several key actions were taken in order to produce a robust model. In order to make the data useable, we first cleaned it by correcting missing values, normalising numbers, and formatting categorical data. The dataset includes comprehensive labels for every flow, including timestamps, IP addresses, ports, protocols, and several attack types like Brute Force, DDoS, Heartbleed, Web Attacks, Infiltration, and Botnet. The dataset spans five days of network activity.

Table 3. Network Flow Features

Feature	Description
totlen_bwd_pkts	Total length of backward packets
flow_pkts_s	Number of packets per second in the flow
tot_bwd_pkts	Total number of backward packets
fwd_pkt_len_mean	Mean length of forward packets
flow_duration	Duration of the flow
cwe_flag_count	Count of CWE (Congestion Window Reduced) flags
fwd_iat_std	Standard deviation of inter-arrival time of forward packets
tot_fwd_pkts	Total number of forward packets
bwd_iat_min	Minimum inter-arrival time of backward packets
psh_flag_cnt	Count of PSH (Push) flags
bwd_iat_std	Standard deviation of inter-arrival time of backward packets
ack_flag_cnt	Count of ACK (Acknowledgment) flags
totlen_fwd_pkts	Total length of forward packets
ece_flag_cnt	Count of ECE (Explicit Congestion Notification Echo) flags
bwd_seg_size_avg	Average size of backward segments
fwd_pkt_len_std	Standard deviation of forward packet length
fwd_iat_max	Maximum inter-arrival time of forward packets
down_up_ratio	Ratio of downloaded to uploaded data
fwd_pkt_len_max	Maximum length of forward packets
urg_flag_cnt	Count of URG (Urgent) flags
pkt_size_avg	Average packet size
bwd_pkt_len_min	Minimum length of backward packets
fwd_iat_min	Minimum inter-arrival time of forward packets

bwd_pkt_len_mean	Mean length of backward packets
bwd_pkt_len_std	Standard deviation of backward packet length
subflow_bwd_byts	Bytes in backward subflow
subflow_bwd_pkts	Packets in backward subflow
fwd_seg_size_avg	Average size of forward segments
bwd_iat_max	Maximum inter-arrival time of backward packets
rst_flag_cnt	Count of RST (Reset) flags
fwd_iat_mean	Mean inter-arrival time of forward packets
subflow_fwd_byts	Bytes in forward subflow
syn_flag_cnt	Count of SYN (Synchronization) flags
bwd_iat_mean	Mean inter-arrival time of backward packets
fwd_pkt_len_min	Minimum length of forward packets

The "Friday-WorkingHours-Afternoon-DDos_aligned" segment alone contains 244,753 labeled records, which provided a substantial amount of data for training and validating our model. To guarantee the generalisability and correctness of the model, the CICIDS2017 dataset was divided into training and testing subsets during the training process. The Random Forest classifier's performance was optimised by hyperparameter tuning, and the model was validated using cross-validation methods. The dataset's extensive traffic diversity and attack variation made it possible for the model to learn and correctly categorise both benign and malicious actions, which has made it extremely useful in real-world scenarios.

As shown in Figure 1, the first 20 lines of the "Friday-WorkingHours-Afternoon-DDos" dataset in CSV format highlight key network flow features such as packet lengths, inter-arrival times, and flag counts. These features were essential in helping the model achieve high accuracy and reliability in detecting intrusions.

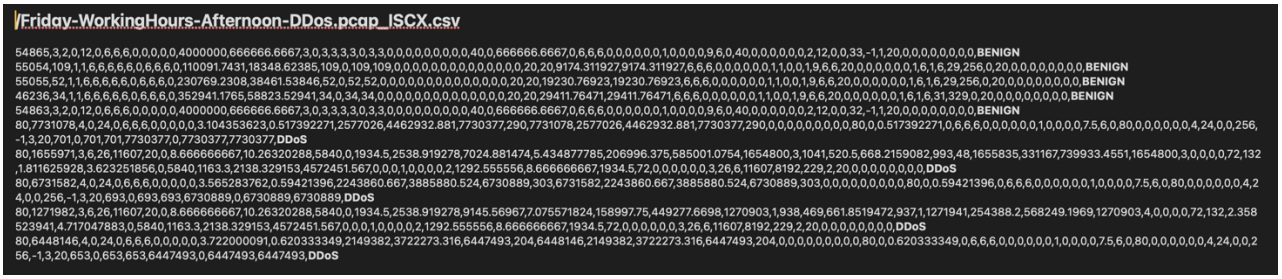


Figure 1. First 20 lines of the "Friday-WorkingHours-Afternoon-DDos" dataset

III.3 Machine Learning Models

In this project, two machine learning models were utilized: the Random Forest Classifier and the Kitsune Autoencoder-Based Framework. These models were selected based on their strengths in handling various aspects of Host Intrusion Detection Systems (HIDS), providing comprehensive and adaptive security measures.

The Random Forest Classifier was selected because of its ability to identify features, handle huge datasets with confidence, and resist overfitting through ensemble learning. This approach offers high classification accuracy for known attack types by constructing numerous decision trees and averaging their results. According to research, random forest models analyse network traffic data well and achieve high detection accuracy and fewer false positives in HIDS applications (Mohi-ud-din et al., 2022). Furthermore, new research suggests that evolutionary algorithms can be used to optimise Random Forest models in order to improve classification performance (Shah et al., 2021). In this study, the CICIDS2017 dataset—a large dataset with realistic background traffic and a variety of contemporary attack types—was used to train the Random Forest Classifier. Studies utilizing this dataset have confirmed the model's effectiveness in classifying network traffic with high accuracy and precision (Chindove & Brown, 2021).

Because it makes use of unsupervised learning, the Kitsune Autoencoder-Based Framework is especially well-suited for identifying new or zero-day attacks. This framework monitors network traffic using several autoencoders, and it exploits reconstruction errors to find anomalies. Kitsune can continuously adapt to changing dangers because it doesn't require labelled training data. Many research have shown how effective autoencoder-based models are at detecting anomalies. For example, Dao and Lee (2022) show how stacked autoencoders can reduce computational complexity and increase detection rates—particularly in contexts with limited resources. Furthermore, it has been demonstrated that integrating autoencoders with random forest models can improve intrusion detection even further, increasing both detection speed and accuracy (Wang et al., 2023).

IV. Implementation

IV.1 Libraries & Frameworks

To successfully implement the chosen machine learning models and facilitate smooth deployment, a selection of libraries and frameworks was utilized. Below, we present an overview of the key tools employed, categorized by their roles in data preprocessing, model training, evaluation, and deployment. Each tool was carefully selected to optimize performance, streamline development, and ensure scalability.

Table 4.Machine Learning Libraries

Library/Framework	Purpose	Version	Description
Scikit-learn	Data Mining and Analysis	1.4.2	Provides simple and efficient tools for data mining and data analysis, including algorithms for classification, regression, clustering, and more.
Pandas	Data Structures and Analysis	2.2.2	Offers powerful data structures and data analysis tools for handling structured data seamlessly. Its Data Frame query capability is central to data manipulation and analysis.
NumPy	Scientific Computing	1.26.4	The foundation for scientific computing in Python, supporting large, multi-dimensional arrays and matrices along with a collection of mathematical functions to operate on these arrays.
SciPy	Technical Computing	1.13.0	Builds on NumPy by providing additional functionality for scientific and technical computing, including modules for

			optimization, integration, interpolation, eigenvalue problems, and others.
Matplotlib & Seaborn	Data Visualization	3.8.4 & 0.13.2	Used for creating static, animated, and interactive visualizations in Python. Seaborn, built on top of Matplotlib, offers a high-level interface for drawing attractive statistical graphics.

The most important Machine Learning Libraries are highlighted in Table 4, including tools for data mining, analysis, scientific computing, and data visualization.

Table 5.Data Processing Libraries

Library/Framework	**Purpose**	**Version**	**Description**
FastParquet & PyArrow	Data Processing	2024.2.0 & 16.0.0	Facilitate reading and writing Apache Parquet files, a columnar storage file format optimized for large-scale data processing.
Category Encoders	Machine Learning	2.6.3	Provides several methods for encoding categorical variables for machine learning algorithms.
PyNTM	Network Topology Modeling	3.4.1	Used for network topology modeling and visualization, enabling detailed analysis and visualization of network structures.

Table 5 provides a selection of libraries essential for efficient data processing, machine learning preparation, and network topology modeling.

Table 6.Web Development Tools

Library/Framework	Purpose	Version	Description
Flask	Web Application Framework	3.0.3	A lightweight WSGI web application framework in Python, designed with simplicity and flexibility in mind, making it a popular choice for web development.
Streamlit	Data Science App Framework	1.35.0	An open-source app framework specifically designed for machine learning and data science projects, allowing for the quick creation and deployment of interactive web applications.
Dash	Web Applications	2.17.0	A productive framework for building web applications in Python, especially suited for data visualization apps with complex interactive user interfaces.
Plotly	Graphing Library	5.21.0	Enables the creation of interactive, publication-quality charts. Integrates seamlessly with Dash, providing a wide range of chart types.
Folium **&** Streamlit-folium	Geospatial Data Visualization	0.16.0 & 0.20.0	Facilitate the creation and integration of interactive maps into Streamlit applications, enhancing geospatial data visualization.

Key frameworks for developing interactive and data-driven web applications are highlighted in Table 6, including tools for visualizing complex data and geospatial information.

Table 7.Utility Libraries

Library/Framework	Purpose	Version	Description
BeautifulSoup4	Web Scraping	4.12.3	A library for web scraping purposes, used to extract data from HTML and XML files.
Requests	HTTP Library	2.31.0	A simple and elegant HTTP library for Python, built for human beings, making it easy to send HTTP requests and handle responses.
SQLAlchemy	SQL Toolkit and ORM	1.4.41	A SQL toolkit and Object-Relational Mapping (ORM) library for Python, providing a full suite of persistence patterns and database interaction capabilities.
Psutil	System Monitoring	5.9.8	A cross-platform library for retrieving information on running processes and system utilization, essential for monitoring and managing system resources.

Utility libraries that support essential tasks like web scraping, HTTP communication, database interaction, and system monitoring in software development are listed in Table 7.

Table 8.Development Tools

Library/Framework	Purpose	Version	Description
IPython	Interactive Computing	8.23.0	An enhanced interactive Python shell offering many features that improve interactive computing.
Jupyter	Interactive Computing Platform	1.0.0	A web-based interactive computing platform that allows users to create and share documents containing live code, equations, visualizations, and narrative text.

Debugpy	Debugging	1.8.1	A powerful debugger for Python, providing a comprehensive suite of interactive debugging capabilities.

Table 8 details tools that enhance the development workflow, including interactive computing environments and debugging tools critical for research and development.

IV.3 Random Forest Model Training

Using the CIC-IDS2017 dataset, the code below describes the precise procedures followed to build a Random Forest classifier for identifying Distributed Denial of Service (DDoS) attacks. Data preparation, model training, assessment, and analysis are all steps in the process. Several machine learning libraries and Python were used for the implementation.
This machine learning model was implemented in a Google Colab notebook, which made it easy to access the necessary libraries and datasets and allowed for smooth execution. The pictures below show the procedures that were used to train this model.

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import
BaggingClassifier,
RandomForestClassifier
from sklearn.metrics import
accuracy_score, recall_score,
precision_score, f1_score,
confusion_matrix, matthews_corrcoef
from sklearn.model_selection import
train_test_split
import joblib
import os
```

Figure 2. Importing essential libraries for data handling, visualization, model training, and evaluation

The implementation, as shown in Figure 2, began by importing essential libraries for data manipulation, visualization, and machine learning, including Pandas for data handling, Seaborn and Matplotlib for visualization, NumPy for numerical operations, Scikit-learn for machine learning and evaluation, Joblib for model saving, and OS for file operations.

```
pd.set_option('display.max_columns',
None)
pd.set_option('display.max_rows',
None)
```

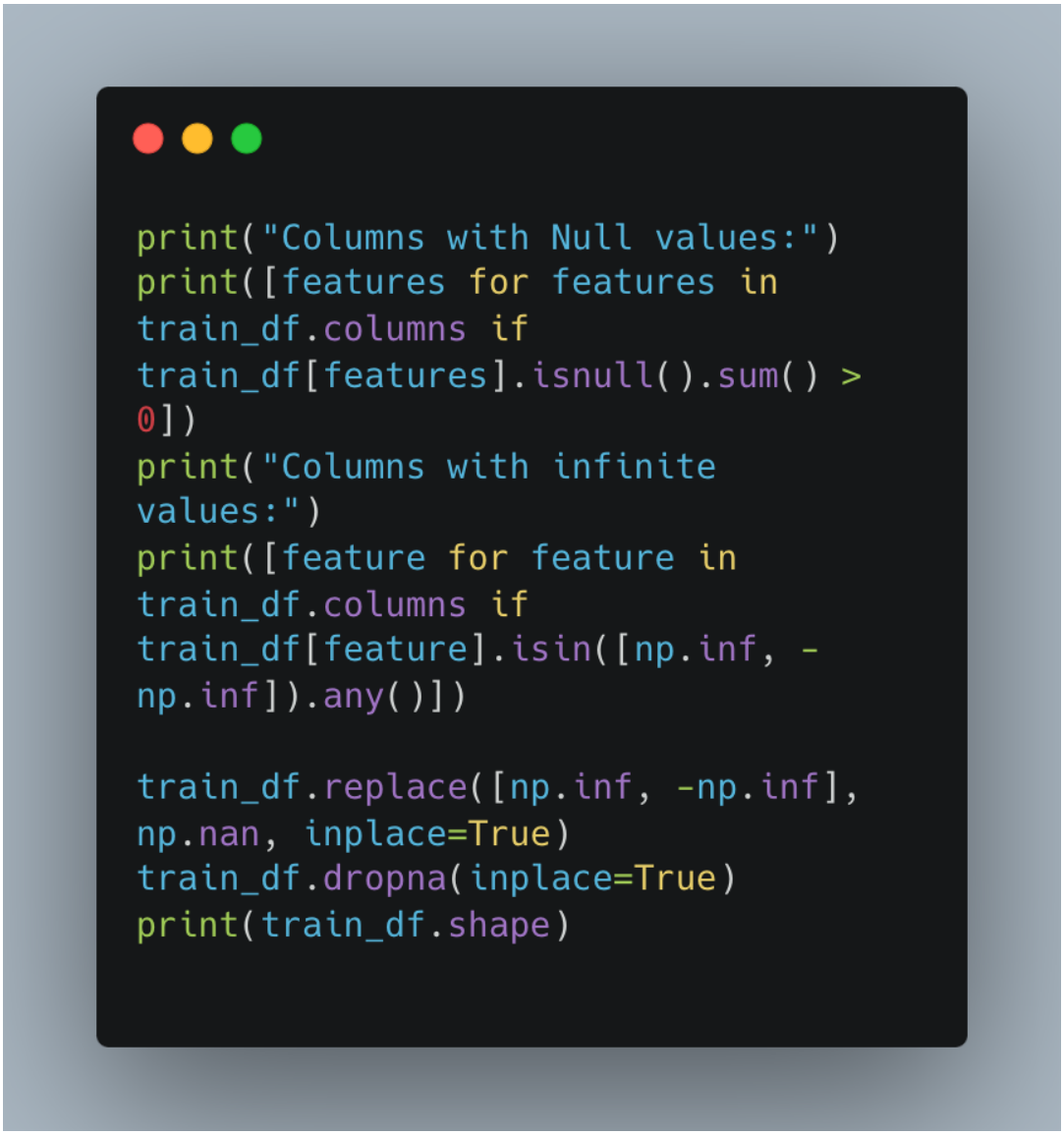
Figure 3. Configuring pandas to display all columns and rows without truncation

The code sets display.max_columns and display.max_rows to None, as shown in Figure 3, to ensure all data is visible during exploration, allowing for a complete view of the dataset without truncation.

```
train_path =
"/content/drive/MyDrive/Colab-
Notebooks/all important/Friday-
WorkingHours-Afternoon-
DDos_aligned.csv"
if not os.path.exists(train_path):
    raise FileNotFoundError(f"File not
found: {train_path}")
train_df = pd.read_csv(train_path)

train_df.info()
print(train_df.keys())
print("Number Of target/Label Column
and their Counts")
print(train_df['Label'].value_counts()
print(train_df.shape)
```

Figure 4. Loading the training dataset from a specified path and displaying basic information about it

The dataset used in this project was the CIC-IDS2017, specifically a subset focused on Friday afternoon DDoS attacks. As shown in Figure 4, the dataset was loaded from a specified path, and it was ensured that the file existed. To facilitate comprehensive data exploration, Pandas display options were set to show all columns and rows.



```
print("Columns with Null values:")
print([features for features in
train_df.columns if
train_df[features].isnull().sum() >
0])
print("Columns with infinite
values:")
print([feature for feature in
train_df.columns if
train_df[feature].isin([np.inf, -
np.inf]).any()])

train_df.replace([np.inf, -np.inf],
np.nan, inplace=True)
train_df.dropna(inplace=True)
print(train_df.shape)
```

Figure 5. Checking for and handling null and infinite values in the dataset

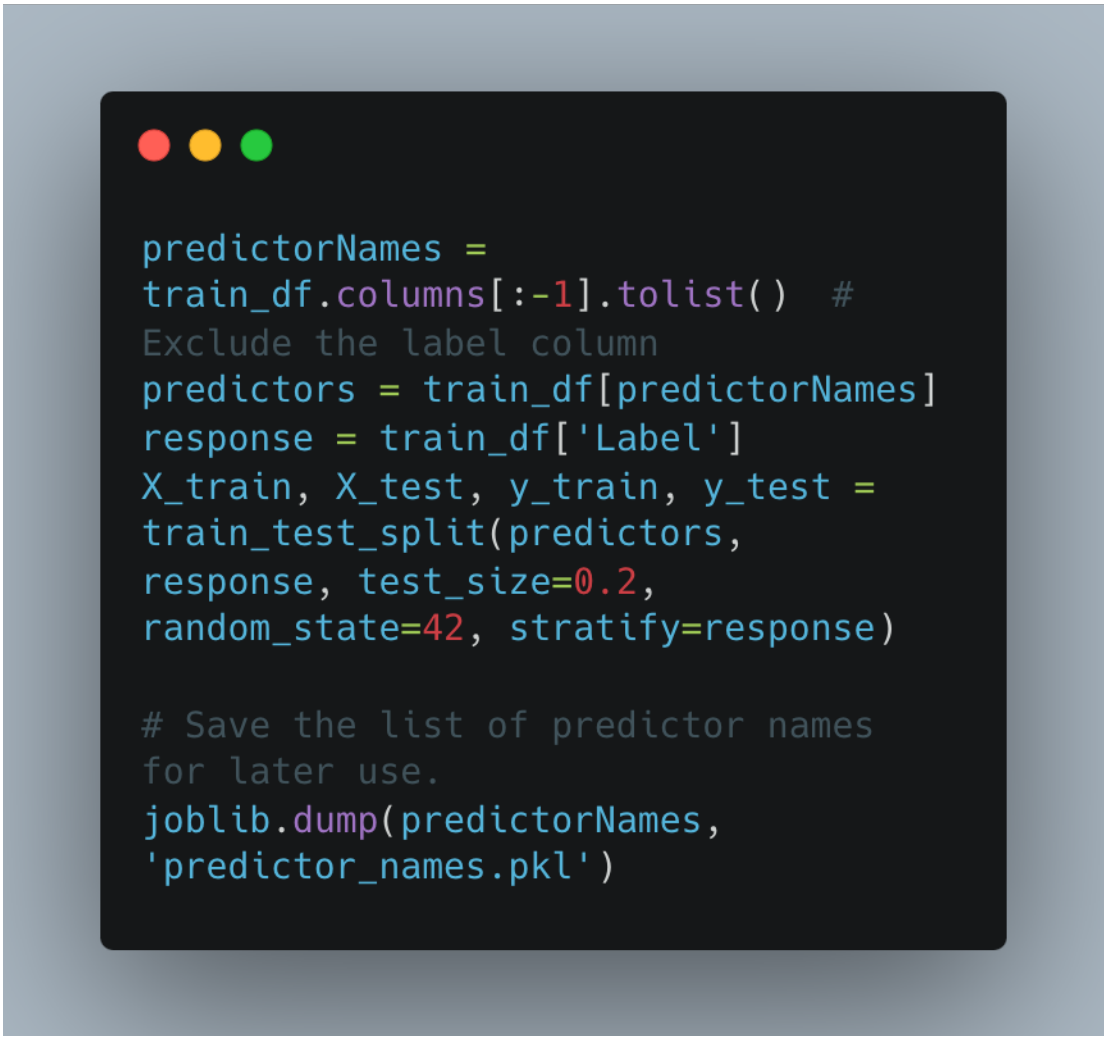
The dataset was checked for missing and infinite values, as illustrated in Figure 5. Infinite values were replaced with NaN, and rows containing NaN values were removed to maintain data integrity.

Figure 6. Visualizing the distribution of the target labels in the dataset

A count plot was created to visualize the distribution of the target labels in the dataset, as shown in Figure 6, displaying the frequency of each label in the Label column to assess class imbalance.

Figure 7. Converting categorical labels to numerical labels for model training

The categorical labels were converted into numerical values for model training, as demonstrated in Figure 7. Specifically, 'BENIGN' was mapped to 0 and 'DDoS' to 1, a crucial step to ensure the machine learning model could accurately process the labels.



```
predictorNames =
train_df.columns[:-1].tolist()  #
Exclude the label column
predictors = train_df[predictorNames]
response = train_df['Label']
X_train, X_test, y_train, y_test =
train_test_split(predictors,
response, test_size=0.2,
random_state=42, stratify=response)

# Save the list of predictor names
for later use.
joblib.dump(predictorNames,
'predictor_names.pkl')
```

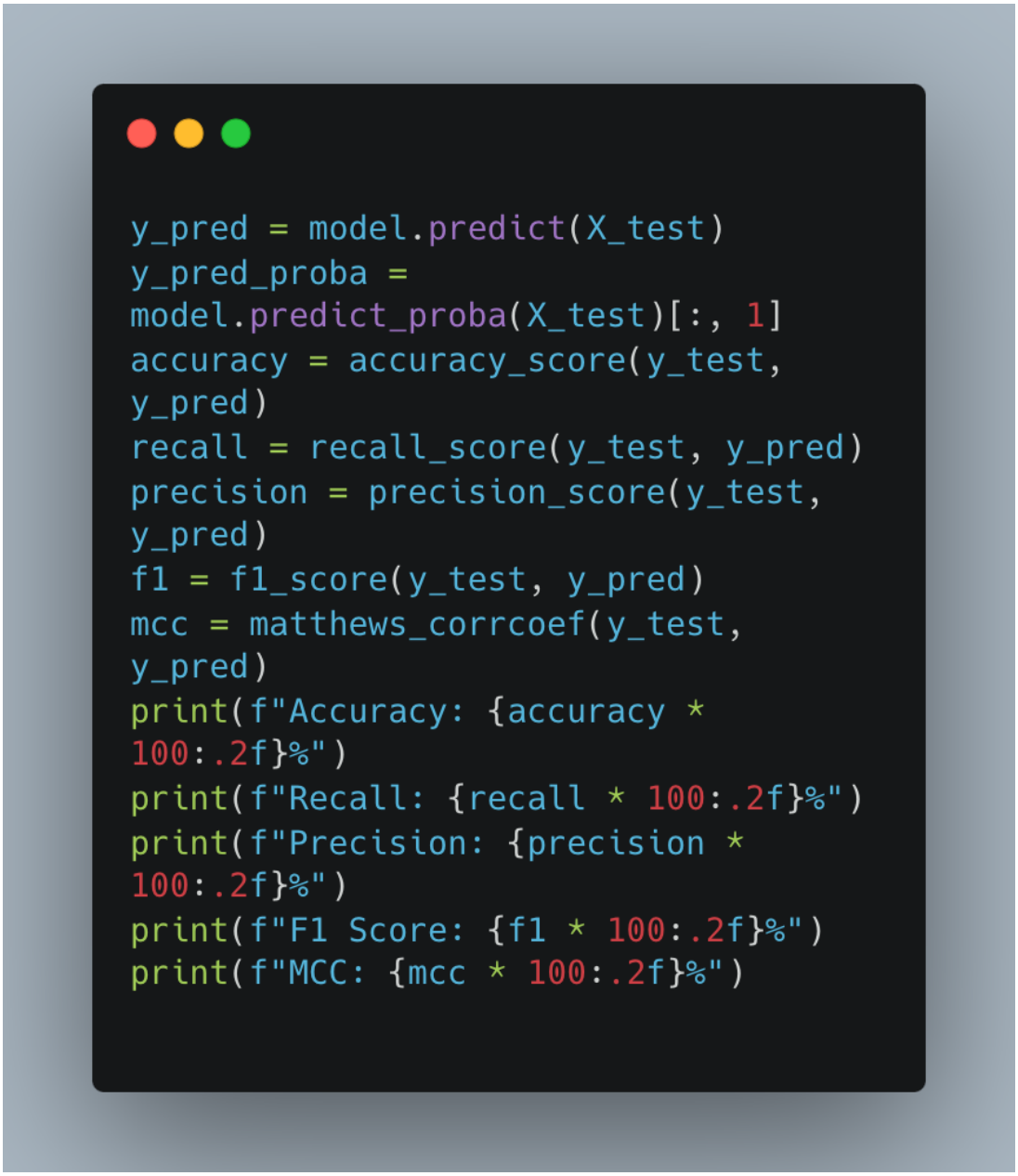
Figure 8. Splitting the dataset into training and testing sets and saving the predictor names

The dataset was split into training and test sets with an 80-20 split ratio, as illustrated in Figure 8. Stratified sampling was employed to maintain the class distribution in both sets, and the predictor names were saved for later use.

```
rfc =
RandomForestClassifier(n_estimators=50
, max_depth=3,
class_weight='balanced')
BaggingClassifier(estimator=rfc,
n_estimators=100, max_samples=0.5,
max_features=0.5)
model.fit(X_train, y_train)

# Save the trained model.
joblib.dump(model, 'best_model.pkl')
```

Figure 9. Initializing and training a Bagging Classifier with a Random Forest base estimator

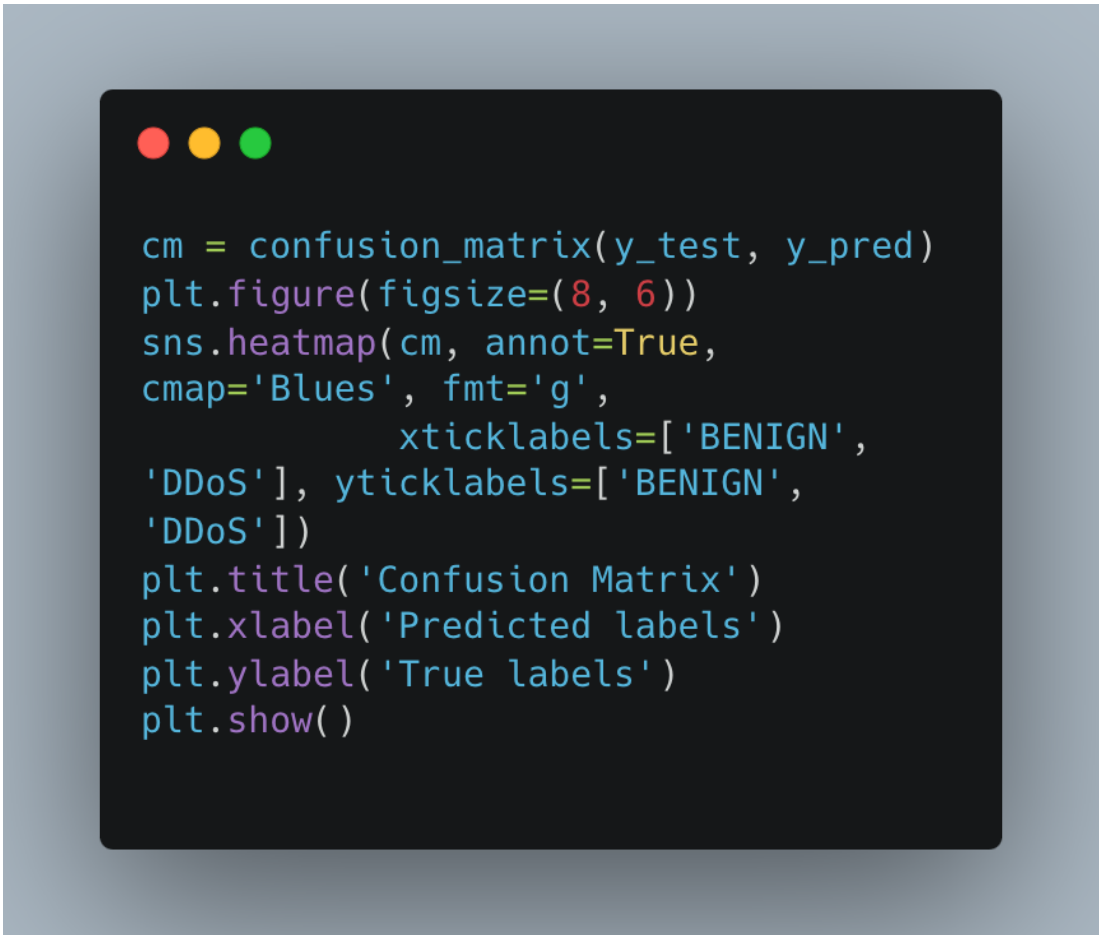
A Random Forest classifier was configured with balanced class weights and trained using a Bagging ensemble method, as shown in Figure 9, to enhance its robustness. The RandomForestClassifier was set up with 50 estimators, a maximum depth of 3, and balanced class weights, and was then used as the base estimator in a BaggingClassifier with 100 base estimators. The trained model was subsequently saved for future use.



```
y_pred = model.predict(X_test)
y_pred_proba =
model.predict_proba(X_test)[:, 1]
accuracy = accuracy_score(y_test,
y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test,
y_pred)
f1 = f1_score(y_test, y_pred)
mcc = matthews_corrcoef(y_test,
y_pred)
print(f"Accuracy: {accuracy *
100:.2f}%")
print(f"Recall: {recall * 100:.2f}%")
print(f"Precision: {precision *
100:.2f}%")
print(f"F1 Score: {f1 * 100:.2f}%")
print(f"MCC: {mcc * 100:.2f}%")
```

Figure 10. Evaluating the trained model on the test set and displaying various performance metrics

The model's performance was evaluated using various metrics, including accuracy, recall, precision, F1 score, and Matthews Correlation Coefficient (MCC), as depicted in Figure 10. Predictions were made on the test set, and these metrics were calculated to assess the model's effectiveness in classifying DDoS attacks.



```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True,
cmap='Blues', fmt='g',
            xticklabels=['BENIGN',
'DDoS'], yticklabels=['BENIGN',
'DDoS'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

Figure 11. Visualizing the confusion matrix to understand the classification performance in more detail

A confusion matrix was plotted, as shown in Figure 11, to visualize the classification performance. This matrix offered a clear representation of the model's ability to distinguish between BENIGN and DDoS instances, highlighting both correctly and incorrectly classified examples.

The images below depict the steps followed in the implementation to test this machine learning model.

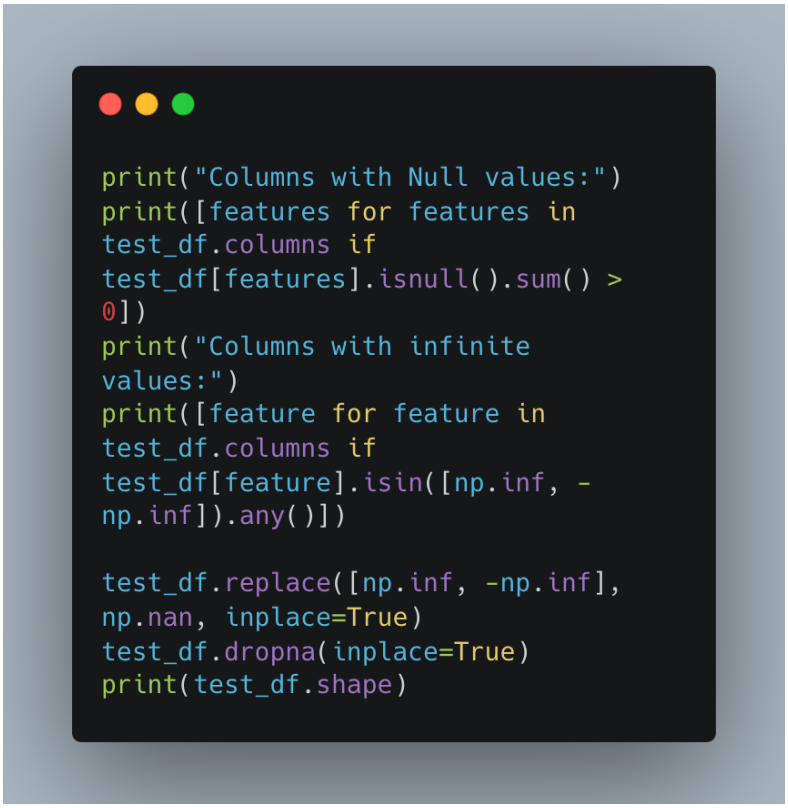
Figure 12. Importing essential libraries for data handling and loading the saved model

The initial step involved importing essential libraries for data manipulation, numerical operations, model loading, and file operations, as shown in Figure 12. Pandas was used for data manipulation, Numpy for numerical operations, Joblib for model loading, and OS for file operations.

```
test_path =
"/content/drive/MyDrive/Colab
Notebooks/all important/Bad-DDOS-
cleaned.csv"
if not os.path.exists(test_path):
    raise FileNotFoundError(f"File
not found: {test_path}")
test_df = pd.read_csv(test_path)

test_df.info()
print(test_df.keys())
print(test_df.shape)
```

Figure 13. Loading the testing dataset from a specified path and displaying basic information about it

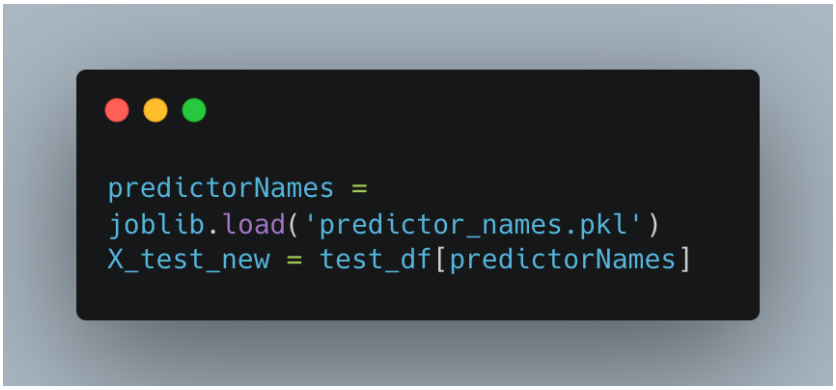
The test dataset, specifically cleaned and prepared for evaluating DDoS attacks, was loaded from a specified path, as depicted in Figure 13. The code verified the file's existence before reading it into a Pandas DataFrame. Initial exploration of the test dataset included displaying its structure, column names, and overall shape to understand its content and structure.



```
print("Columns with Null values:")
print([features for features in
test_df.columns if
test_df[features].isnull().sum() >
0])
print("Columns with infinite
values:")
print([feature for feature in
test_df.columns if
test_df[feature].isin([np.inf, -
np.inf]).any()])

test_df.replace([np.inf, -np.inf],
np.nan, inplace=True)
test_df.dropna(inplace=True)
print(test_df.shape)
```

Figure 14. Checking for and handling null and infinite values in the testing dataset

The test dataset was checked for missing and infinite values, which were handled appropriately. As illustrated in Figure 14, infinite values were replaced with NaN, and rows containing NaN values were removed to ensure data integrity, a crucial step to prevent errors during the model's prediction phase.



```
predictorNames =
joblib.load('predictor_names.pkl')
X_test_new = test_df[predictorNames]
```

Figure 15. Preparing the test data by ensuring it has the same predictors as the training data

To ensure consistency with the training phase, the same predictor names (features) used during training were loaded. As shown in Figure 15, these predictors were then used to create the feature set for the test data, ensuring that the model's input format matched that of the training data.

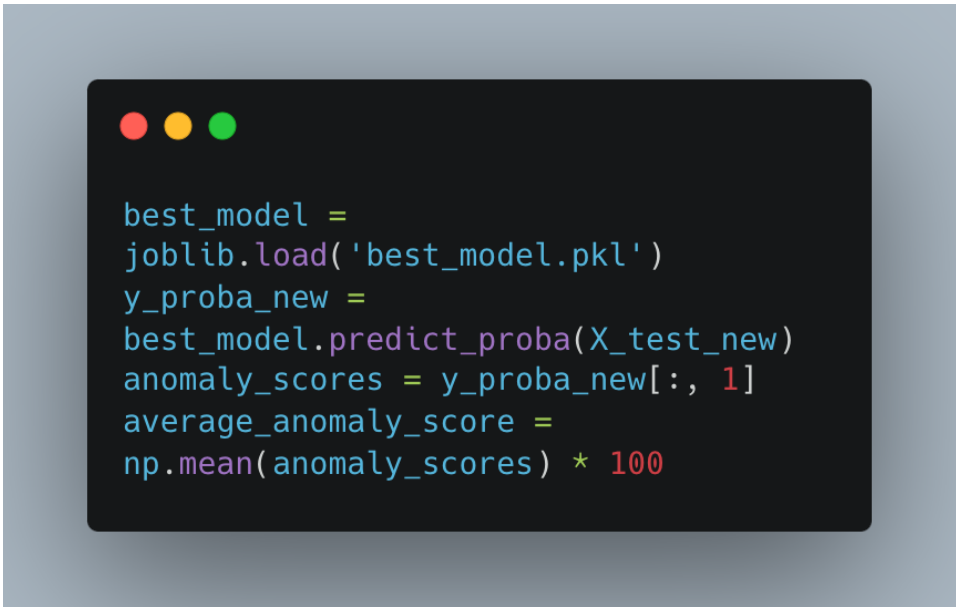
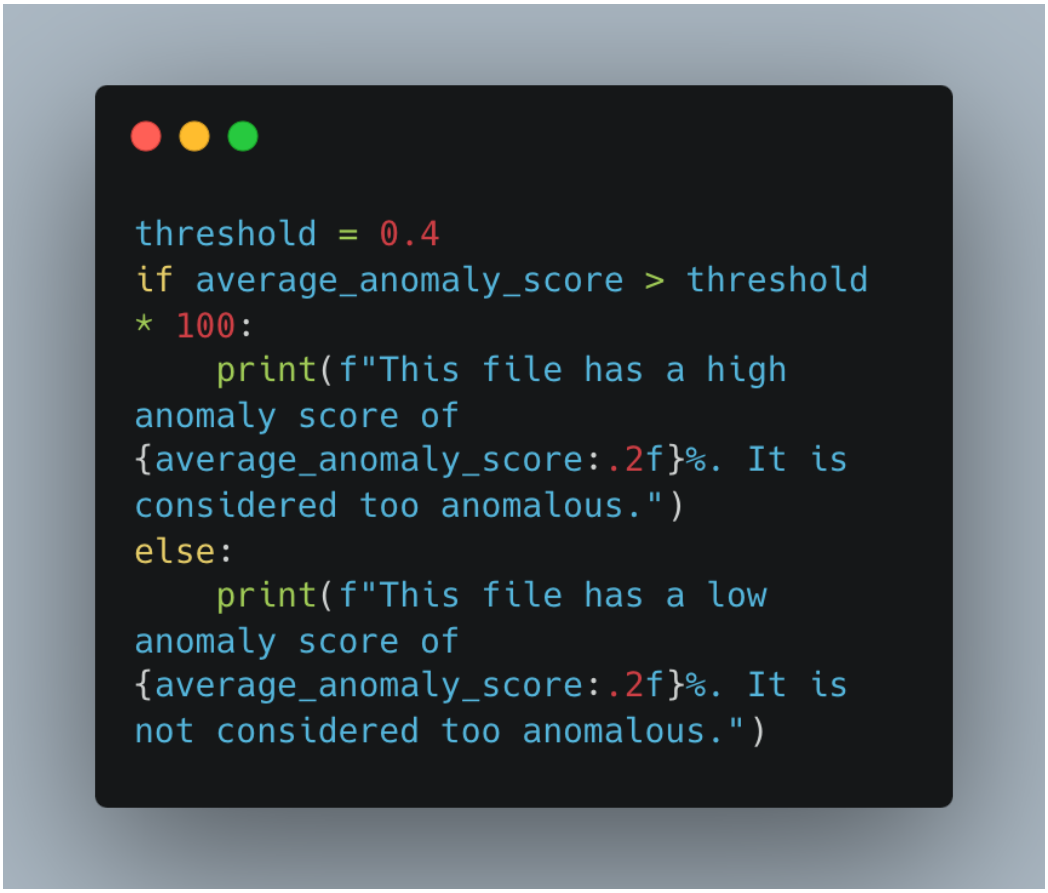


Figure 16. Loading the saved model and making predictions on the new testing data

The trained model was loaded from a saved file, and predictions were made on the test dataset using this model. As depicted in Figure 16, the model's prediction probabilities were used to calculate anomaly scores, indicating the likelihood of each instance being an attack. This step provided a detailed understanding of how likely each test instance was to be classified as an anomaly (attack).



```
threshold = 0.4
if average_anomaly_score > threshold
* 100:
    print(f"This file has a high
anomaly score of
{average_anomaly_score:.2f}%. It is
considered too anomalous.")
else:
    print(f"This file has a low
anomaly score of
{average_anomaly_score:.2f}%. It is
not considered too anomalous.")
```

Figure 17. Evaluating the anomaly score of the test data to determine if it is considered anomalous

An anomaly score threshold was set to classify the test file as either anomalous (high likelihood of attacks) or normal. The average anomaly score of the test dataset was calculated and compared against this threshold. As illustrated in Figure 17, if the average anomaly score exceeded the threshold, the test file was classified as highly anomalous, indicating a significant likelihood of containing DDoS attacks. Conversely, if the average anomaly score was below the threshold, the test file was considered to have a low likelihood of containing anomalies.

IV.4 Kitsune

Kitsune is a framework for network-based anomaly detection that makes use of the KitNET ensemble of autoencoders. This study's Kitsune implementation combines a number of frameworks and modules to effectively detect and visualise network anomalies. Streamlit is the primary user interface library; Pandas is used for data management; Matplotlib is used for charting; Scipy is used for statistical analysis; and Numpy is used for numerical calculations. The Kitsune module is also essential for handling network packets and identifying irregularities.

Implementation Details

The implementation is demonstrated in the images below.

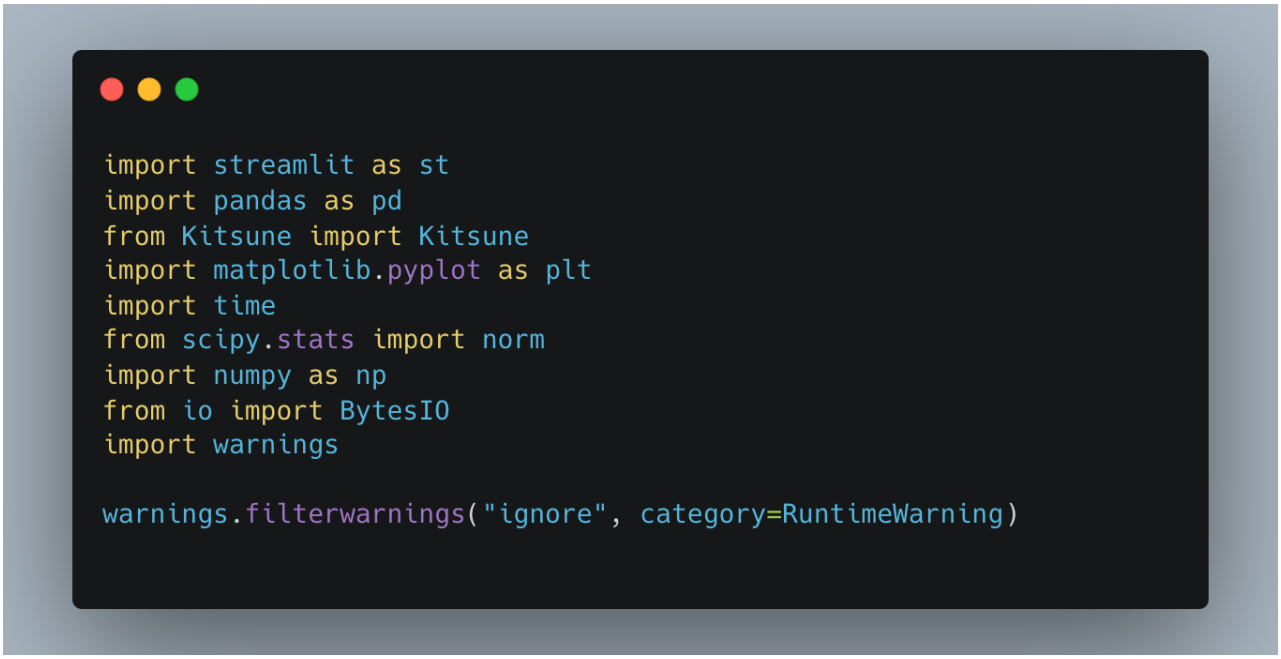
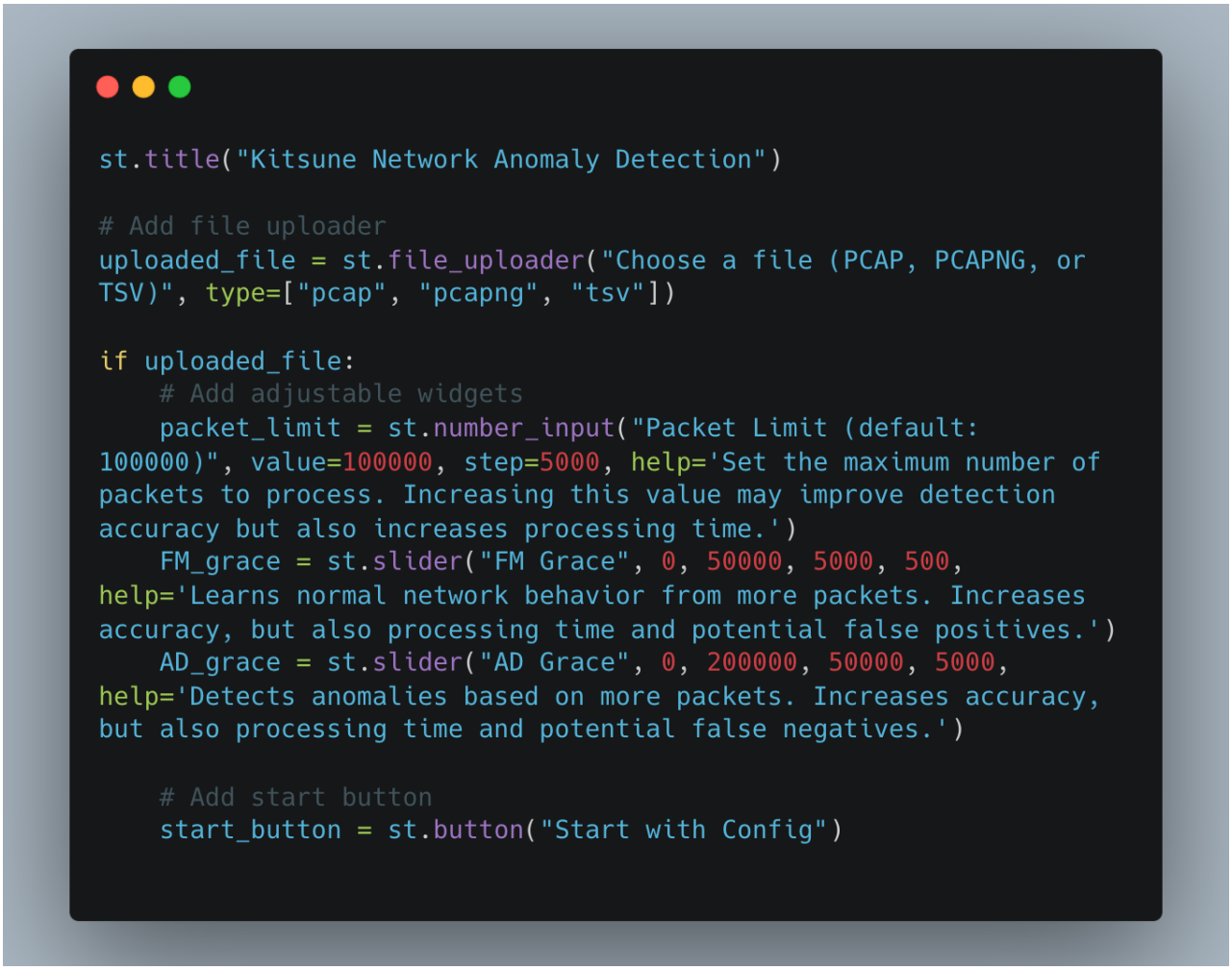


Figure 18. Python Code Import Statements

The necessary Python libraries were imported for data manipulation, visualization, statistical analysis, and the execution of the Kitsune algorithm. Streamlit was imported for creating web apps, Pandas for data handling, Matplotlib for plotting, and SciPy for statistical functions. Additionally, warnings related to runtime issues were suppressed to avoid unnecessary console output, as depicted in Figure 18.



```
st.title("Kitsune Network Anomaly Detection")

# Add file uploader
uploaded_file = st.file_uploader("Choose a file (PCAP, PCAPNG, or
TSV)", type=["pcap", "pcapng", "tsv"])

if uploaded_file:
    # Add adjustable widgets
    packet_limit = st.number_input("Packet Limit (default:
100000)", value=100000, step=5000, help='Set the maximum number of
packets to process. Increasing this value may improve detection
accuracy but also increases processing time.')
    FM_grace = st.slider("FM Grace", 0, 50000, 5000, 500,
help='Learns normal network behavior from more packets. Increases
accuracy, but also processing time and potential false positives.')
    AD_grace = st.slider("AD Grace", 0, 200000, 50000, 5000,
help='Detects anomalies based on more packets. Increases accuracy,
but also processing time and potential false negatives.')

    # Add start button
    start_button = st.button("Start with Config")
```

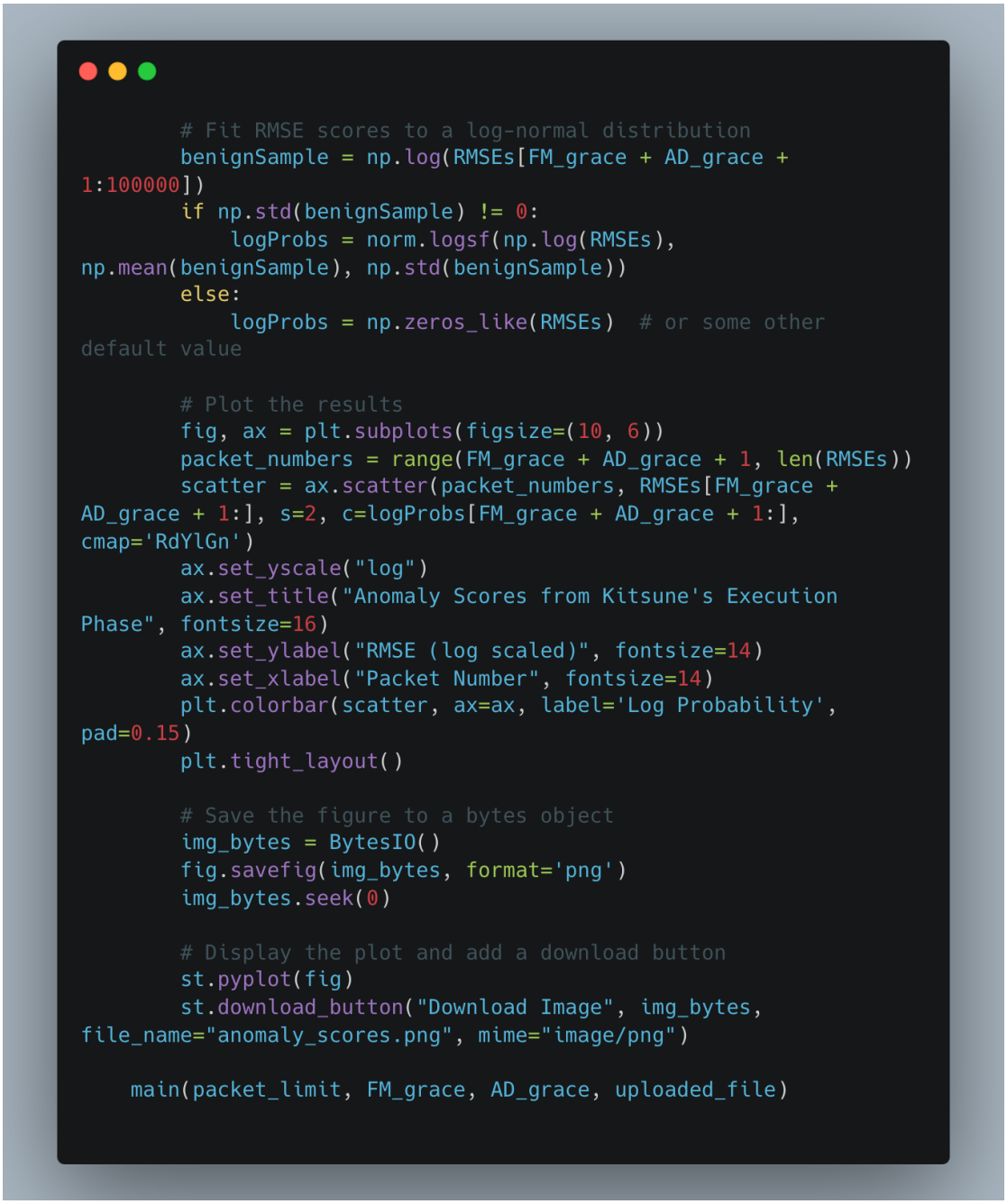
Figure 19. Streamlit App Function for Kitsune NID

Streamlit is used to create a user-friendly web interface where users can upload network capture files (PCAP, PCAPNG, or TSV). The interface includes adjustable widgets to configure the parameters for the Kitsune anomaly detection process, such as packet limits and grace periods for the Feature Mapper (FM) and Anomaly Detector (AD), as shown in Figure 19.

```
if start_button:
    def main(packet_limit, FM_grace, AD_grace, uploaded_file):
        # Process the uploaded file with Kitsune
        file_path = uploaded_file.name
        K = Kitsune(file_path, packet_limit)
        st.write(f"Running Kitsune on {file_path}")

        RMSEs = []
        timestamps = []
        i = 0
        start_time = time.time()

        # Progress bar
        total_packets = packet_limit if packet_limit < float('inf')
else None
        progress_bar = st.progress(0)
        packets_processed_text = st.text("Packets processed: 0")

        while True:
            rmse = K.proc_next_packet()
            if rmse == -1:
                break
            RMSEs.append(rmse)
            timestamps.append(time.time() - start_time)
            i += 1

            # Update packets processed display every 1000 packets
            if i % 1000 == 0:
                packets_processed_text.text(f"Packets processed:
{i}")

            # Update progress bar
            if total_packets is not None:
                progress_bar.progress(i / total_packets)

        # Close the progress bar
        progress_bar.empty()
        st.success("Kitsune execution completed.")
```

Figure 20. Main Function for Kitsune NID

Upon starting the detection process, the main function initializes the Kitsune instance with the specified parameters and processes the uploaded file. The function also manages the progress of packet processing, updates the progress bar, and displays the number of packets processed, as shown in Figure 20.



```
        # Fit RMSE scores to a log-normal distribution
        benignSample = np.log(RMSEs[FM_grace + AD_grace +
1:100000])
        if np.std(benignSample) != 0:
            logProbs = norm.logsf(np.log(RMSEs),
np.mean(benignSample), np.std(benignSample))
        else:
            logProbs = np.zeros_like(RMSEs)  # or some other
default value

        # Plot the results
        fig, ax = plt.subplots(figsize=(10, 6))
        packet_numbers = range(FM_grace + AD_grace + 1, len(RMSEs))
        scatter = ax.scatter(packet_numbers, RMSEs[FM_grace +
AD_grace + 1:], s=2, c=logProbs[FM_grace + AD_grace + 1:],
cmap='RdYlGn')
        ax.set_yscale("log")
        ax.set_title("Anomaly Scores from Kitsune's Execution
Phase", fontsize=16)
        ax.set_ylabel("RMSE (log scaled)", fontsize=14)
        ax.set_xlabel("Packet Number", fontsize=14)
        plt.colorbar(scatter, ax=ax, label='Log Probability',
pad=0.15)
        plt.tight_layout()

        # Save the figure to a bytes object
        img_bytes = BytesIO()
        fig.savefig(img_bytes, format='png')
        img_bytes.seek(0)

        # Display the plot and add a download button
        st.pyplot(fig)
        st.download_button("Download Image", img_bytes,
file_name="anomaly_scores.png", mime="image/png")

    main(packet_limit, FM_grace, AD_grace, uploaded_file)
```

Figure 21. RMSE Score Plotting and Download Function

The implementation further includes fitting the Root Mean Squared Error (RMSE) scores to a log-normal distribution and plotting the results using Matplotlib. Figure 21 shows how the plot visualizes the anomaly scores across packets processed, aiding in the interpretation of detection results.

IV.4 Deploying in VPS

In this section, we detail the deployment of a Streamlit application to a Virtual Private Server (VPS) hosted on a Proxmox server. The VPS configuration, featuring 8 cores, 8 GB of RAM, and 100 GB of storage, offers enhanced control and flexibility compared to managed services.

The deployment process starts with configuring the VPS, as demonstrated in the images below.

Figure 22. Command to Update and Upgrade System Packages

After accessing the VPS via SSH, the system is updated, and the necessary packages are installed to ensure the environment is up-to-date and ready for further configuration, as shown in Figure 22.

Figure 23. Commands to Create and Enable Swap File

Creating a 2 GB swap space is essential for optimizing memory usage and preventing application crashes caused by insufficient RAM, as illustrated in Figure 23.

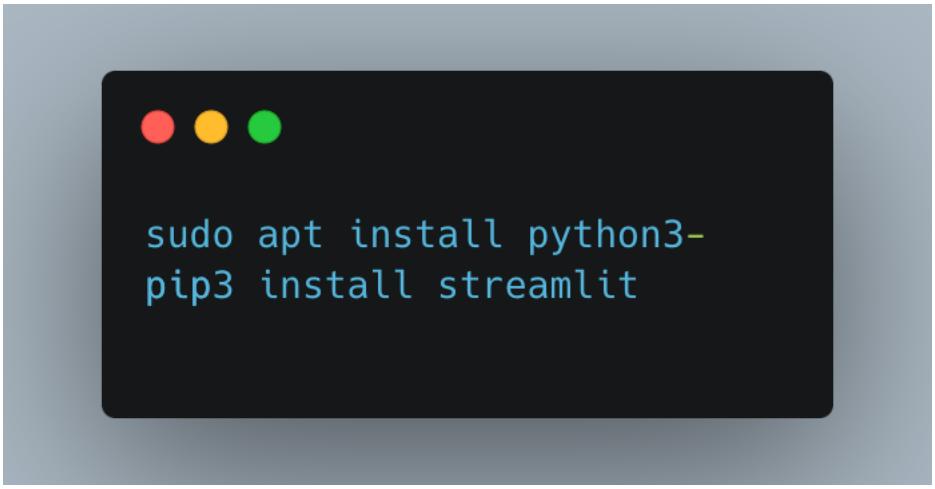


Figure 24. Commands to Install Python3 and Streamlit

To support the Streamlit application, both pip for Python 3 and Streamlit are installed, as demonstrated in Figure 24.

Figure 25. Command to Add a New User for Streamlit

For security purposes, a dedicated user is created to run the Streamlit application, reducing the risk of system-wide issues, as shown in Figure 25.

The application files are transferred from the local machine to the VPS using FileZilla, an FTP client. This secure file transfer process ensures that the files are correctly placed in the home directory of the Streamlit user on the VPS (/home/streamlit/streamlit-app).

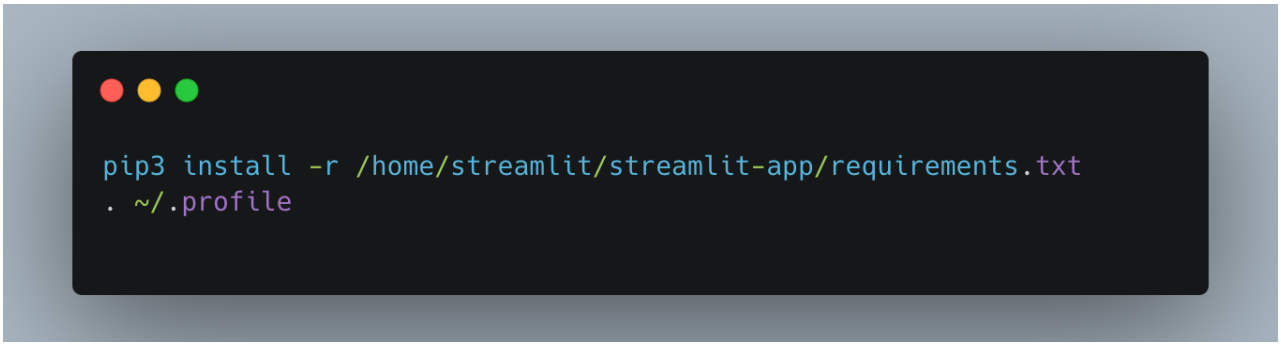


Figure 26. Command to Install Requirements for Streamlit App and Source Profile

If the application requires additional Python packages, these dependencies are specified in a requirements.txt file and installed accordingly, as shown in Figure 26.

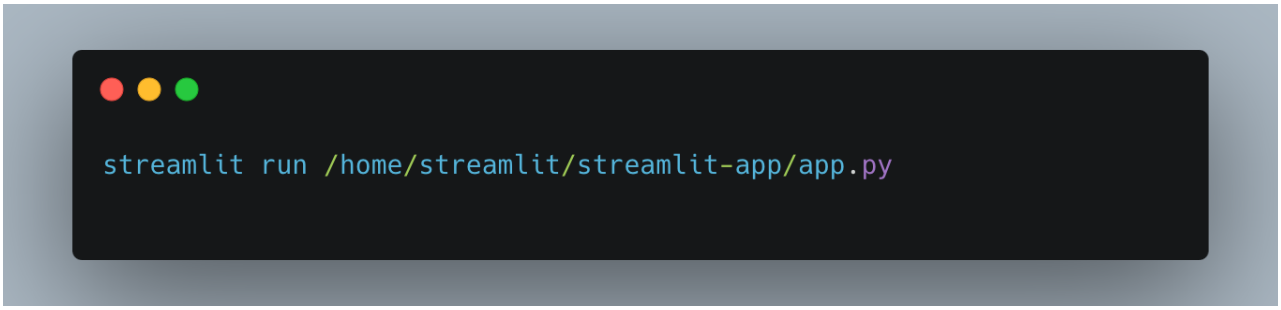


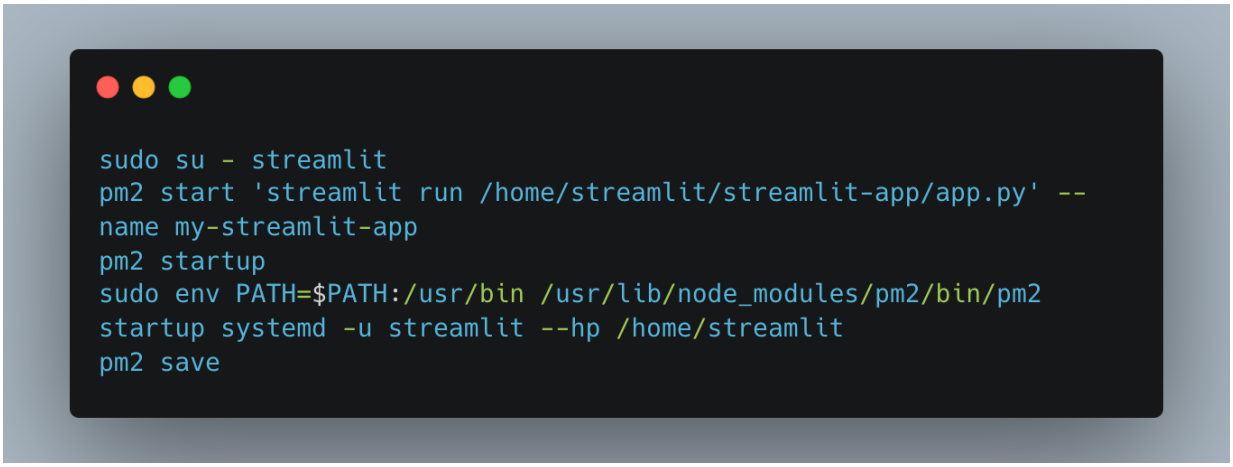
Figure 27. Command to Run Streamlit App

Launching the Streamlit application on the VPS requires running a command that initiates the app at a specified local URL, as shown in Figure 27.

```
sudo apt-get update && sudo apt-get install -y ca-certificates curl
gnupg
curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-
repo.gpg.key | sudo gpg --dearmor -o
/etc/apt/keyrings/nodesource.gpg
NODE_MAJOR=20
echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg]
https://deb.nodesource.com/node_$NODE_MAJOR.x nodistro main" | sudo
tee /etc/apt/sources.list.d/nodesource.list
sudo apt-get update && sudo apt-get install nodejs -y
sudo npm install pm2@latest -g
```

Figure 28. Commands to Install Node.js and PM2

To ensure the application runs continuously and is efficiently managed, PM2, a process manager for Node.js applications, is used. This process includes installing Node.js, npm, and then PM2, as shown in Figure 28.



```
sudo su - streamlit
pm2 start 'streamlit run /home/streamlit/streamlit-app/app.py' --
name my-streamlit-app
pm2 startup
sudo env PATH=$PATH:/usr/bin /usr/lib/node_modules/pm2/bin/pm2
startup systemd -u streamlit --hp /home/streamlit
pm2 save
```

Figure 29. Commands to Start Streamlit App with PM2 and Enable Auto-Startup

PM2 is then employed to start the Streamlit app and manage it as a background service, ensuring continuous operation even if the terminal session is closed or the VPS is rebooted, as illustrated in Figure 29.

Through these detailed steps, the Streamlit application is successfully deployed to a VPS, achieving efficient operation and effective management.

V. Results & Analysis

V.1 Interpretation and Discussion of Results

In this section, we will interpret the results of two machine learning models: a Random Forest Classifier and an Autoencoder-based framework, each designed to detect host intrusions using the CICIDS2017 dataset.

V.1.1 Random Forest Classifier

The label distribution chart in Figure 30 illustrates a balanced dataset, with a slightly higher number of DDoS instances compared to benign traffic. This balance is vital for our Random Forest model's performance, as it ensures the model has sufficient data to effectively learn from both classes.

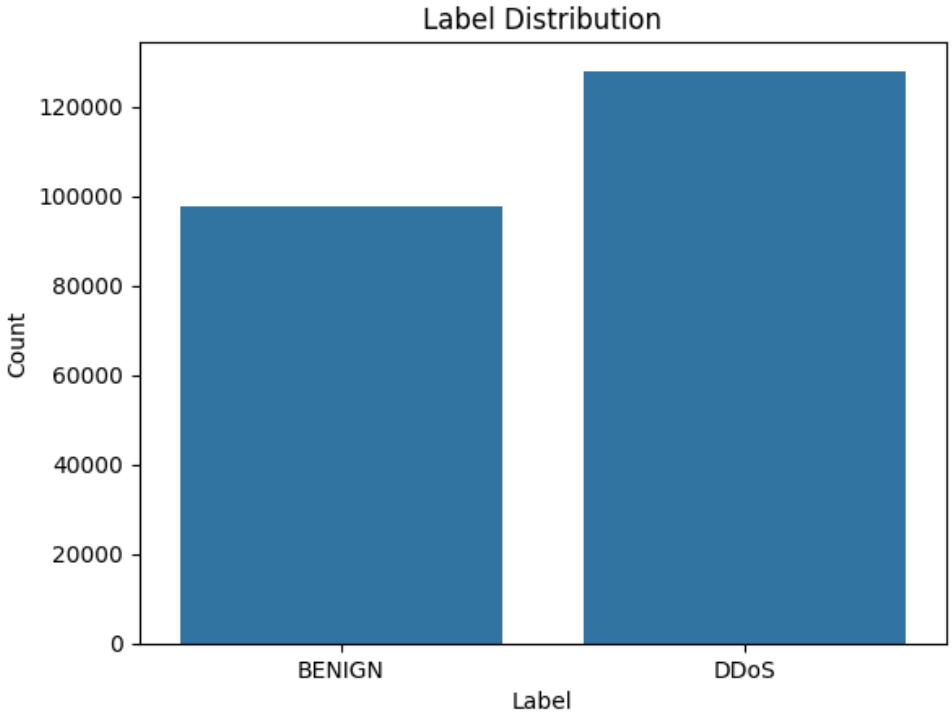


Figure 30. Label Distribution Bar Chart for BENIGN and DDoS

The Random Forest model's confusion matrix shows how well it can differentiate between DDoS and benign data. The model performs robustly, with 19,489 true positives for benign traffic and 25,574 true positives for DDoS. The incredibly low counts of false positives (49) and false negatives (31) show how reliable the model is in detecting known network assaults. The Random Forest model

performs exceptionally well because it can effectively handle enormous datasets and carry out feature selection.

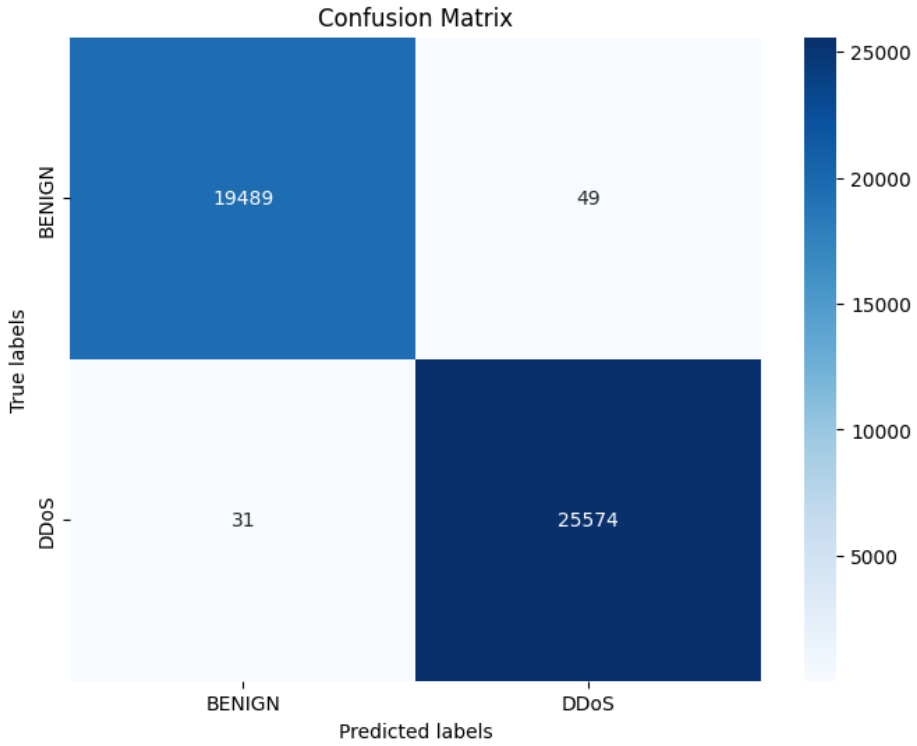


Figure 31. Confusion Matrix for BENIGN and DDoS Classification

Random Forest Classifier's performance on the CICIDS2017 dataset is shown in Table 9.

Table 9.Performance Metrics Table for CICIDS2017 Dataset

Metric	Accuracy	Recall	Precision	F1 Score	MCC
%	99.68%	99.88%	99.56%	99.72%	99.35%

V.1.2 Autoencoder-Based Framework

The confusion matrix of the Random Forest model in Figure 31 demonstrates its high accuracy in distinguishing between benign and DDoS traffic. With 19,489 true positives for benign traffic and 25,574 true positives for DDoS, the model shows robust performance. The low counts of false

positives (49) and false negatives (31) indicate the model's reliability in identifying known network attacks. This performance highlights the Random Forest model's strength in handling large datasets and efficiently performing feature selection.

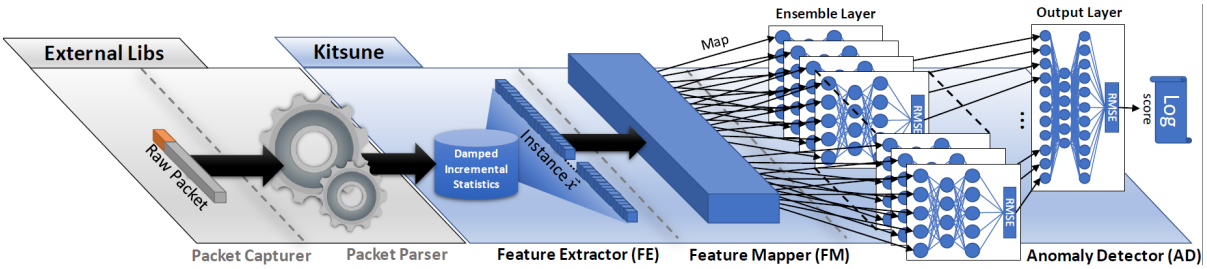


Figure 32. Kitsune Network Anomaly Detection Architecture (Mirsky, Doitshman, Elovici, & Shabtai, 2018)

The Kitsune Network Anomaly Detection Architecture (Mirsky, Doitshman, Elovici, & Shabtai, 2018), as seen in Figure 32, offers a complementary advantage by identifying novel anomalies through its autoencoder-based framework in an unsupervised learning manner. While the Random Forest model is highly effective for detecting known attacks due to its supervised learning approach. The two models are essential components of an all-encompassing network security approach because they combine the abilities to identify known and new threats.
The ensuing illustrations proficiently exhibit anomaly scores produced throughout Kitsune's implementation stage, providing a graphical depiction of network traffic irregularities spanning three distinct datasets.

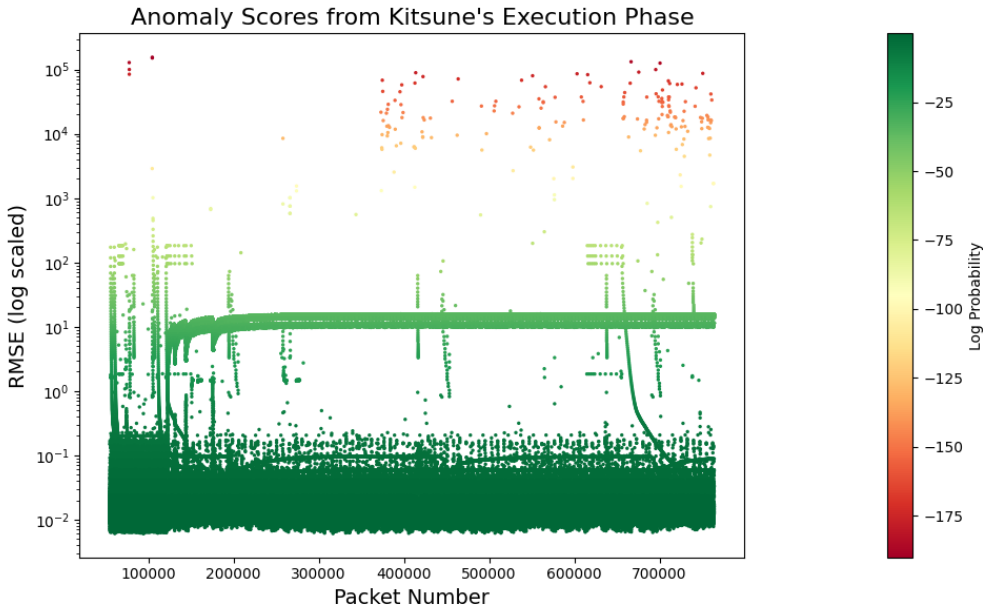


Figure 33. Anomaly Scores from Kitsune's Execution Phase (First Dataset)

In Figure 33, anomaly scores are plotted for the first dataset processed by Kitsune during the analysis of the DarkWave.pcap file, which contains approximately 800,000 packets captured over 118 minutes during a network infection by the Mirai botnet malware. Kitsune used 5,000 instances to learn the feature mapping (FM Grace) and 50,000 instances to train the anomaly detector (AD Grace). The RMSE values, scaled logarithmically, indicate the severity of anomalies, with higher scores and red colors highlighting more significant issues.

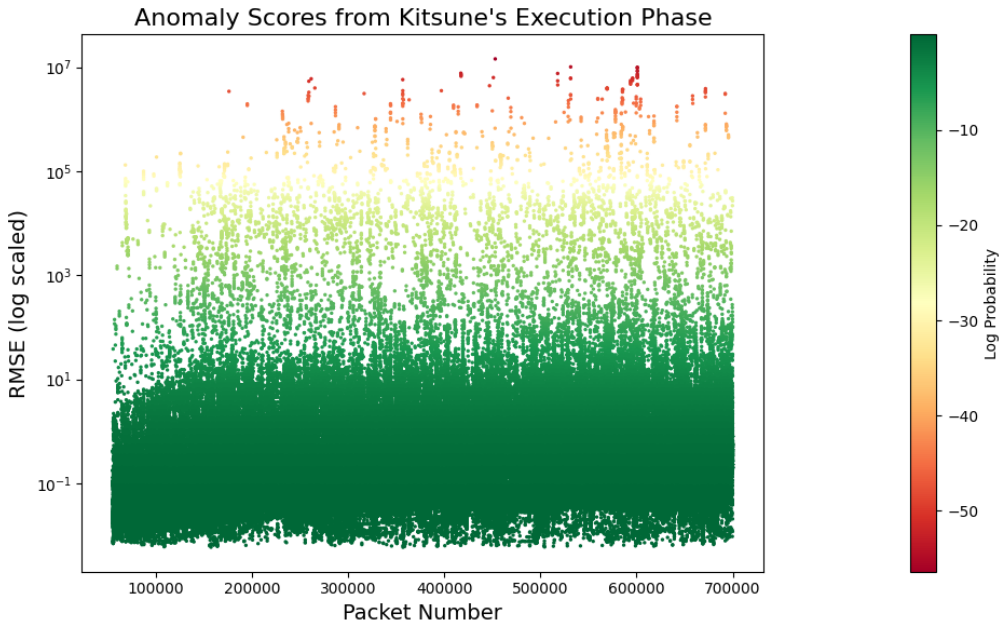


Figure 34. Anomaly Scores from Kitsune's Execution Phase (Second Dataset)

Figure 34 illustrates the anomaly scores from the bigFlows.pcap file, which contains nearly 700,000 packets captured over 5 minutes during a network infection by OS Scanning. Kitsune again used 5,000 instances to learn the feature mapping (FM Grace) and 50,000 instances to train the anomaly detector (AD Grace), with all packets included in the analysis. The RMSE values, shown on a logarithmic scale, reveal significant anomalies, marked by clusters of high scores in red.

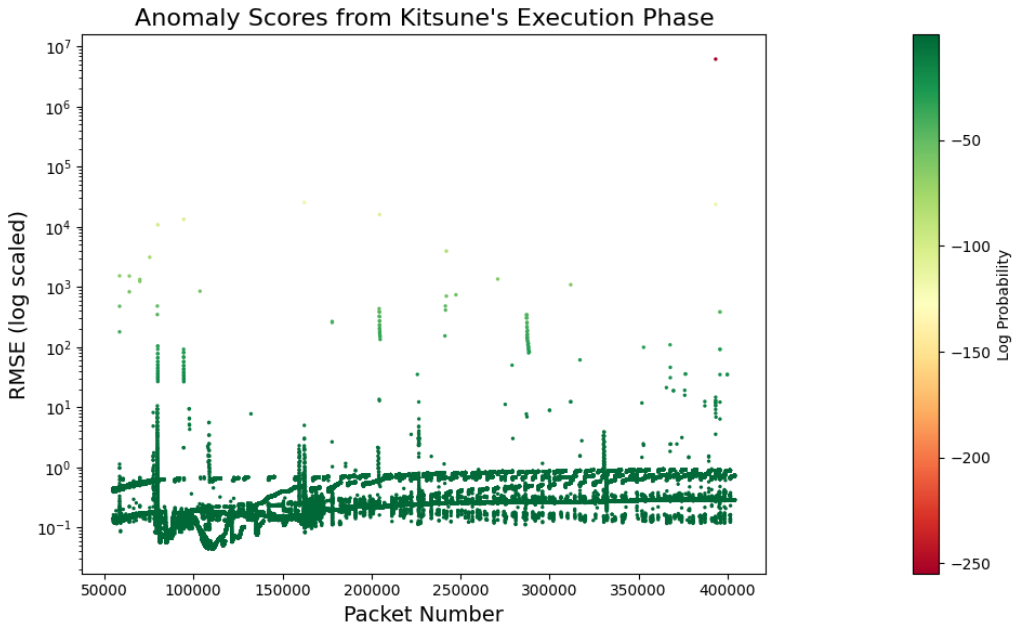


Figure 35. Anomaly Scores from Kitsune's Execution Phase (Third Dataset)

In Figure 35, anomaly scores are plotted for the third dataset, processed by Kitsune during the analysis of the home-400k.pcap file, which contains 400,000 packets captured over 7 minutes during normal home network activity. As with the previous datasets, Kitsune used 5,000 instances for FM Grace and 50,000 instances for AD Grace, with all packets included in the analysis. The RMSE values, displayed on a logarithmic scale, reflect the overall low anomaly level, with only one red dot indicating a minor anomaly.

These visualizations underscore the effectiveness of Kitsune's ensemble of autoencoders in continuously learning and adapting to the normal network traffic behavior, making it highly suitable for anomaly detection in dynamic network environments.

Conclusion

In this work, there are applied and assessed two different machine learning techniques for network anomaly detection: an Autoencoder-based framework called Kitsune and a Random Forest Classifier. The main objective was to incorporate these models into an intuitive web application with a focus on effectiveness, usability, and visual clarity. This important feature guarantees that anyone, even without technical experience, may utilise the tool for network security and understand it. The online application's layout and execution enable users to effortlessly upload network traffic data, adjust detection parameters, and view the outcomes, making sophisticated anomaly detection approachable to a wider range of users.

The Random Forest Classifier turned out to be a very good option for identifying DDoS attack patterns and assessing the attack's likelihood with percentage-based accuracy. Its capacity to efficiently handle both small and large datasets, along with its strong performance in spotting known network assaults, makes it a dependable tool for routine network monitoring. But the classifier's vulnerability to zero-day attacks—new and unidentified threats that it hasn't been trained to detect—is limited by its reliance on training data. The Random Forest Classifier, which offers quick and precise detection for known threats, is nevertheless an important part of our integrated detection system in spite of this drawback.

Kitsune framework was used in this online application to combat the problem of zero-day assaults. It is an excellent unsupervised learner thanks to its group of autoencoders, which allows it to identify new abnormalities with a high degree of accuracy. This system is especially useful for detecting zero-day attacks since it continuously learns the typical behaviour of network traffic and highlights deviations. Complementing the Random Forest Classifier, Kitsune offers subtle and detailed anomaly detection, ensuring thorough coverage of recognised as well as unexpected dangers. When combined, these models offer a strong, multifaceted approach to network security that improves our anomaly detection system's overall efficacy.

Bibliography

Abbas, S. H., Naser, W. A. K., & Kadhim, A. A. (2023). Intrusion detection system (IDS) and intrusion prevention system (IPS). Global Journal of Engineering and Technology Advances. https://doi.org/10.30574/gjeta.2023.14.2.0031

Akhtar, M., & Feng, T. (2022). Malware analysis and detection using machine learning algorithms. Symmetry, 14(11), Article 2304. https://doi.org/10.3390/sym14112304

Al Jallad, K., Aljnidi, M., & Desouki, M. S. (2020). Anomaly detection optimization using big data and deep learning to reduce false-positive. Journal of Big Data, 7(1), 1-12. https://doi.org/10.1186/s40537-020-00346-1

Al Waili, A. R. (2023). Analysis of traffic using the Snort tool for the detection of malware traffic. International Journal of Information Technology and Computer Science, 33(1), 30-37. https://doi.org/10.55529/ijitc.33.30.37

Bhadauria, S., & Mohanty, T. (2021). Hybrid intrusion detection system using an unsupervised method for anomaly-based detection. 2021 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), 1-6. https://doi.org/10.1109/ANTS52808.2021.9936919

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM Computing Surveys, 41(3), Article 15. https://doi.org/10.1145/1541880.1541882

Chindove, H., & Brown, D. (2021). Adaptive machine learning-based network intrusion detection. Proceedings of the International Conference on Artificial Intelligence and Its Applications. https://doi.org/10.1145/3487923.3487938

Coulibaly, K. (2020). An overview of intrusion detection and prevention systems. arXiv. https://arxiv.org/abs/2004.08967

Dao, T. N., & Lee, H. (2022). Stacked autoencoder-based probabilistic feature extraction for on-device network intrusion detection. IEEE Internet of Things Journal, 9(16), 14438-14451. https://doi.org/10.1109/JIOT.2021.3078292

Erlacher, F., & Dressler, F. (2022). On high-speed flow-based intrusion detection using Snort-compatible signatures. IEEE Transactions on Dependable and Secure Computing, 19(1), 495-506. https://doi.org/10.1109/tdsc.2020.2973992

Jouini, M., Ben Arfa Rabai, L., & Ben Aissa, A. (2014). Classification of security threats in information systems. Procedia Computer Science, 32, 489-496. https://doi.org/10.1016/j.procs.2014.05.452

Kumar, S., Gupta, S., & Arora, S. (2021). Research trends in network-based intrusion detection systems: A review. IEEE Access, 9, 148567-148597. https://doi.org/10.1109/ACCESS.2021.3129775

Liao, H.-J., Lin, C.-H. R., Lin, Y.-C., & Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. Journal of Network and Computer Applications, 36(1), 16-24. https://doi.org/10.1016/j.jnca.2012.09.004

Mirsky, Y., Doitshman, T., Elovici, Y., & Shabtai, A. (2018). Kitsune: An ensemble of autoencoders for online network intrusion detection. arXiv. https://arxiv.org/abs/1802.09089

Mohi-ud-din, G., Zheng, J., Liu, Z., Asim, M., Chen, J., Liu, J., & Lin, Z. (2022). NIDS: Random forest-based novel network intrusion detection system for enhanced cybersecurity in VANETs. 2022 International Conference on Virtual Reality, Human-Computer Interaction and Artificial Intelligence. https://doi.org/10.1109/VRHCIAI57205.2022.00051

Rele, M., & Patil, D. (2023). Intrusive detection techniques utilizing machine learning, deep learning, and anomaly-based approaches. 2023 IEEE International Conference on Cryptography,

Informatics, and Cybersecurity (ICoCICs), 88-93. https://doi.org/10.1109/ICoCICs58778.2023.10276955

Shah, S., Muhuri, P. S., Yuan, X., Roy, K., & Chatterjee, P. (2021). Implementing a network intrusion detection system using semi-supervised support vector machine and random forest. Proceedings of the 2021 ACM Southeast Conference. https://doi.org/10.1145/3409334.3452073

Sommer, R., & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. In 2010 IEEE Symposium on Security and Privacy (pp. 305-316). IEEE. https://doi.org/10.1109/SP.2010.25

Ullah, I., Raza, B., Ali, S., Abbasi, I. A., Baseer, S., & Irshad, A. (2021). Software-defined network-enabled fog-to-things hybrid deep learning-driven cyber threat detection system. Security and Communication Networks. https://doi.org/10.1155/2021/6136670

Wang, C., Sun, Y., Wang, W., & Liu, H. (2023). Hybrid intrusion detection system based on a combination of random forest and autoencoder. Symmetry. https://doi.org/10.3390/sym15030568