Anomaly Detection System

Minor project report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology

in

Information Technology

By

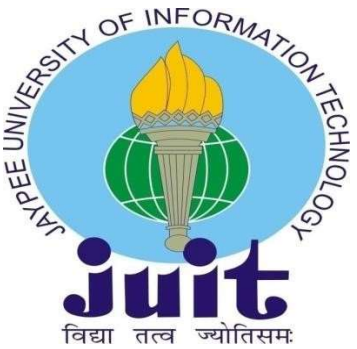
AATISH SHARMA (201536)

ANIMESH (201567)

UNDER THE SUPERVISION OF

Dr. AMOL VASUDEVA

ASSISTANT PROFESSOR (SG.)



Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, 173234,

Himachal Pradesh, INDIA.

TABLE OF CONTENT

LIST OF FIGURES

DECLARATION

I hereby declare that this project has been done by me under the supervision of **Dr. Amol Vasudeva, Assistant professor (SG.)**, Jaypee University of Information Technology. I also declare that neither this project nor any part of this project has been submitted elsewhere for award of any degree or diploma.

Supervised by:

Dr. Amol Vasudeva,

Assistant professor (SG.),

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology

Submitted by:

Aatish Sharma

201536

Animesh

201567

Computer Science & Engineering Department

Jaypee University of Information Technology

CERTIFICATE

This is to certify that the work which is being presented in the project report titled "Anomaly Detection System" in partial fulfilment of the requirements for the award of the degree of B.Tech in Information Technology and submitted to the Department of Computer Science & Engineering And Information Technology, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by "Aatish Sharma (201536) ,Animesh (201567)." during the period from January 2022 to May 2022 under the supervision of Dr.Amol Vasudeva, Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat.

Aatish Sharma

201536

Animesh

201567

The above statement made is correct to the best of my knowledge.

Dr. Amol Vasudeva

Assistant Professor (SG.)

Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat .

ACKNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to almighty God for His divine blessing makes it possible to complete the project work successfully.

I am really grateful and wish my profound indebtedness to the Supervisor **Dr.Amol Vasudeva, Assistant Professor (SG.)** Department of CSE Jaypee University of Information Technology,Wakhnaghat. Deep Knowledge & keen interest of my supervisor in the field of "**Cyber Security**" to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would like to express my heartiest gratitude to **Dr. Amol Vasudeva**, Department of CSE, for his kind help to finish my project.

I would also generously welcome each one of those individuals who have helped me straight forwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patients of my parents.

Aatish Sharma

201536

Animesh

201567

ABSTRACT

The development of a cyber security anomaly detection system is the main goal of this project. By analysing network data and observing abnormal patterns of behaviour, anomaly detection systems are essential for spotting and blocking possible cyber threats. Machine learning techniques are used by the system to identify both known and unknown threats when it comes to these anomalies.

The project involves creating a scalable and effective system that can effectively process massive amounts of data. The system development process involves several crucial processes, including feature extraction from the data and selection. To train the model to recognise and classify various kinds of cyber threats, labelled data was used.

Overall, the anomaly detection system is a valuable tool for enhancing the cyber security and mitigating potential cyber risks. As the threats continue to evolve, it is essential to develop effective and efficient systems to detect and prevent cyber attacks. While this system may not operate in real-time, it still a powerful tool for detecting and preventing cyber threats.

Chapter 01 : Introduction

1.1 Introduction

In today's digital age, cyber security is really important as we use technology more in our daily lives. Cyber attacks and data breaches are becoming more common, so we need to protect sensitive information and prevent damage. Anomaly detection is a way to do this by analyzing network traffic and finding unusual behavior that might be a threat. We need efficient and effective anomaly detection systems because cyber attacks are getting more complex. This project is about making an anomaly detection system that can find different types of cyber threats using machine learning. The system can process lots of data quickly and accurately and can adapt to new threats, which makes it really useful for improving cyber security.

1.2 Objective

This report is about how machine learning can help with cyber security by detecting potential threats. The main aim of an anomaly detection system is to improve cyber security by identifying any unusual activity. By creating a system that can protect against both known and unknown attacks, we can safeguard important information and infrastructure. It's important to have a flexible and adaptable solution that can keep up with the constantly changing threats.

1.3 Motivation

The motivation behind this report to address the importance of anomaly detection in a world where the nature cyber attacks are constantly evolving .It is necessary to understand the methods and tools in the field of cybersecurity in order to keep up with existing methods and their applications. With the rise of AI technologies these methods have also changed for better. This report discusses the use AI and machine learning in anomaly detection of cyber security threats.

Overall, The motivation of this report is to highlight the importance of anomaly detection in cyber security and discuss a methodology of how AI and machine learning can be used in anomaly detection for cybersecurity. This report focuses on a system made for specific protocols ,However, The methodology discussed in the report can be generalised for any protocol or technology.

1.4 Language Used

Python is a commonly used language for machine learning and developing GUI, due to its ease of use.

- Programming Language : Python
- Library : pandas, numpy, scikit learn, tensorflow, custom tkinter ,os

1.5 Technical Requirements

- Desktop or laptop computer with good internet Connection.
- Wireshark or any other network traffic analysis tool and CICflowmeter.
- Minimal RAM

1.6 Outcomes

- The system provides the user with two categories of network data which they can use to identify cyber security threats and perform further information analysis on it to investigate and mitigate any risk.
- The system acts as the first layer of analysis in case of any query related to cyber threats comes to the user.
- The data generated by the system after performing quries can be used for either improving the detection model or generating a new model.

Chapter 02 : Feasibility Study | Requirements Analysis | Design

2.1 Feasibility Study

2.1.0 Literature Review

1.Introduction

Information security is a critical concern of todays society and with the advancement of technology the cyber security has emerged as a field to counter this concern to some extent. However ,The cyber security is highly dependent on the detection of the attacks from the external and internal threats. This has resulted in demand of techniques and methods that are able to detect attacks. Anomaly detection systems have emerged as a promising approach to identifying suspicious activities in large datasets, which can help prevent data breaches and other security incidents. This literature review aims to examine the existing literature on anomaly detection systems for information security. Specifically, this review will focus on the design and implementation of such systems which have emphasis on machine learning approaches for anomaly detection. The review will also evaluate the performance of these systems using real-world data and discuss the challenges and limitations associated with their use.

Our motivation: Using unsupervised machine learning algorithms to detect known as well as unknown web attacks .

Past Works-

(Moradi Vartouni, A., Teshnehlab, M., & Sedighian Kashi, S. ,2019)

In this study the author has leveraged DNN methods to extract relevant features from HTTP request logs for anomaly detection in Web Application Firewalls(WAF). N-gram based on character is a method that They used to construct features from the data. Also, Stacked Autoencoder (SAE)and Deep Belief Network(DBN) are used as Deep Neural Network(DNN) methods, which are applied to 2-gram features, then, They fused them with 1-gram features in parallel to robust resulting feature sets for anomaly detection purposes. They applied detection methods on CSIC 2010 and ECML/PKDD 2007 datasets. 1SVM, elliptic envelope, and IF have been implemented as one-class learner to distinguish normal data from abnormal ones. The results showed that in contrast to the traditional models, deep models especially fusion models have better performance regarding accuracy and generalisation in reasonable detection time.

(Pan, Y., Sun, F., Teng, Z., White, J., Schmidt, D. C., Staples, J., & Krause, L. 2019)

In the paper author Pan et al. (2019), an architecture for unsupervised end-to-end deep learning to detect attacks on web applications is described. The authors examined and kept track of the behaviour of web applications during runtime using the Robust Software Modelling Tool (RSMT). To create a low-dimensional representation of call traces extracted from application runtime data, they used a denoising autoencoder. They developed test

applications and synthetic trace datasets, then assessed unsupervised learning performance against them to validate their intrusion detection system.

Traditional cross-validation is not often used to evaluate deep learning models due to computational costs. To enable a fair comparison with other machine learning methods, cross-validation was not used in their experiments.

The work presented in the paper highlights several key lessons learned, including:

• Based on reconstruction error, autoencoders can learn descriptive representations of web application stack trace data that can distinguish between typical and abnormal requests. Developers are protected from unnecessary details by the learned representation, which highlights important characteristics.

• Without requiring domain knowledge, unsupervised deep learning can identify many attacks, such as SQL injection, XSS, or deserialization, with above 0.91 F1-score.

• End-to-end deep learning outperforms systems created with specialised human knowledge in the detection of web attacks. Without requiring application-specific previous information, the autoencoder-based method outperforms supervised methods in terms of performance.

The authors' work highlights the potential of unsupervised deep learning in web application security without the extensive domain knowledge or labeled training data.

(Ghazal, S. F., & Mjlae, S. A.,2022)

Ghazal et al. (2022) evaluated eleven different classification techniques on HTTP DATASET CSIC 2010 dataset, including three deep learning approaches [long short-term memory, convolutional neural networks, and long short-term memory + convolutional neural networks] [Decision Tree, Random Forest, Gradient Boosting, XGBoost, AdaBoost, Multilayer Perceptron, and Voting]. They used these techniques to identify suspicious assaults. The most straightforward performance metric used in the study was the correctly predicted sample ratio. The dataset was transformed from text to numbers using the Label Encoder method. The algorithms were evaluated using the f1-score, along with precision, accuracy, and recall. The LSTM with CNN outperformed the other models in terms of detecting assaults, with scores of 0.995, 1.0, 0.99, and 0.995 for accuracy, precision, and recall, respectively. In this test, DT and voting classifiers outperformed the top machine learning methods. The results suggest that a range of assault kinds can be detected using machine learning and deep learning.

Ferriyan, A., Thamrin, A. H., Takeda, K., & Murai, J. ,2021)

Ferriyan et al. (2021) provide a new dataset for benchmarking and benchmarking IDS in this article. They stress the significance of having current datasets, especially given how quickly network traffic changes. Two major contributions are made by the paper. First, the authors present brand-new specifications for developing datasets, such as anonymization, payload,

ground-truth, encryption, and practical implementation techniques, which are not addressed in existing datasets. Second, the authors provide a brand-new, publicly accessible IDS dataset called HIKARI-2021 that includes network traffic with encrypted traces. They categorised each flow as benign or an assault and offer pcap files for all traces. Features like the source IP address, source port, destination IP address, and destination port are included in the dataset.

Previous works with different datasets , algorithms and their results-

References	Year	Attack	Cybersecurity Dataset	Algorithm	Result
Kim, A., Park, M., and Lee, D. H. (2020), "AI-IDS: Application of deep learning to real-time Web intrusion detection", IEEE Access, Vol. 8, pp.70245-70261.	2020	Normal Anomalous	CSIC 20 I 0 dataset	CNN and LSTM DNN	CNN and LSTM accuracy = 91 .54%
Vartouni, A. M., Kashi, S. S., and Teshnehlab, M. (2018), "An anomaly detection method to detect web attacks using stacked auto-encoder". In 2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS), pp. 131-134.	2018	Normal Anomalous	CSJC 20 I 0 dataset	Stacked AutoEncoder	Accuracy = 88.32 %
Betarte, G., Pardo, Á., and Martínez, R. (2018), "Web application attacks detection using machine learning techniques", In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), pp.1065-1072.	2018	Normal Anomalous	CSJC 20 I 0 dataset	Random Forest KNN-3 SVM	Random Forrest accuracy = 72%
Tuan, T. A., Long, H. V., Kumar, R., Priyadarshini, I., and Son, N. T. K. (2019), "Performance evaluation of Botnet DDoS attack detection using machine learning", Evolutionary Intelligence, pp.1-12.	2019	• Analysis •Reconnaissance • DoS • Exploits • Fuzzers • Generic • Normal • Worms • Backdoor • Shellcode	UNSW-NBl5	SVM, ANN, NB, DT ,and USML	Max Accuracy = 0.94
Anwer, M., Farooq, M. U., Khan, S. M., and Waseemullah, W. (2021), "Attack Detection in IoT using Machine Learning", Engineering, Technology and Applied Science Research, Vol. 11 No. 3, pp.72737278.	2021	• R2L ·Dos • U2R • Probe • Normal	NSL-KDD	SVM, GBDT, and RF	Accuracy = 0.95
Su, T., Sun, H., Zhu, J., Wang, S., and Li, Y. (2020), "BAT: Deep learning methods on network intrusion detection using NSL-KDD dataset", IEEE Access, Vol. 8, pp.29575-29585	2020	• R2L • DoS • U2R • Probe • Normal	NSL-KDD	BAT model	Accuracy = 0.96

Xu, W., Jang-Jaccard, J., Singh, A., Wei, Y., and Sabrina, F. (2021), "Improving Performance of Autoencoder-Based Network Anomaly Detection on NSL-KDD Dataset", IEEE Access, Vol. 9, pp.140136-140146	2021	• R2L • DoS • U2R • Probe • Normal	NSL-KDD	5·laycr autoencoder (AE) based model	Accuracy = 0.97
Kavitha, S., and Uma Maheswari, N. (2021), "Network Anomaly Detection for NSL-KDD Dataset Using Deep Learning", Information Technology in Industry, Vol. 9 No. 2, pp.821-827	2021	• R2L • DoS • U2R • Probe • Normal	NSL-KDD	One Class SVM	Accuracy = 0.98
Ferriyan, A., Thamrin, A. H., Takeda, K., and Murai, J. (2021) ,"Generating Network Intrusion Detection Dataset Based on Real and Encrypted Synthetic Attack Traffic", Applied Sciences, Vol. 11 No. 17.	2021	• Background • Benign • Brute force • Probing • XMRJGCC Crypto Miner	ALLFLOWMETER HIKARI2021	KNN, SVM, RF and MLP	Accuracy = 0.99

Conclusion-

A lot of work has been done in the field of anomaly detection and Intrusion detection for cyber security .However, Most of the studies lack real world use case and replicability of the datasets generated, Due to lack of information related to generation of dataset. HIKARI dataset counters this problem to some extent .It is a relatively new dataset and also addresses some recent types of attacks. HIKARI-2021 dataset based on network traffic that is specifically HTTPS protocol and application layer attacks that use HTTPS. The main reason of author for choosing HTTPS protocol is the report from the 2021 Data Breach Investigation, 80% of the attack vectors come from application layer attacks. Retraining ,using online data from real-world usage opens the possibility of incorporating attack data into the normal behavior data set and improving existing accuracy even further. Unsupervised and Deep learning approaches can be a key factor in achieving it. The implementation of these techniques on the HIKARI-2021 dataset is an area of further research.

Literature Review Table

S No.	Year	Title	Published in	Work	Remarks
1	2009[1]	Anomaly Detection : A Survey.	*ACM Computing Surveys*	This reference gives a detailed view of Anomalies and various machine learning techniques of detecting them.	Anomaly detection is a key stake holder in Information security with application in various different fields such as intrusion detection and fraud detection Unsupervised learning is not suitable for bulk anomaly detection
2	2019[2]	Leveraging deep neural networks for anomaly-based web application firewall	*IET Information Security*, *13*(4)	This reference uses various DNN algorithms to detect anomalies in HTTP queries	The IF(Isolation Forest) algorithm and 1-SVM (One Class Support Vector Machine) in SAE(Stacked Auto Encoder) and DBN(Deep Belief Network) perform better for anomaly detection .However, IF preform ideally.
3	2019[3]	Detecting web attacks with end-to-end deep learning	*Journal of Internet Services and Applications*, *10*(1)	This reference explores the deep learning algorithms and the processes associated to detect web attacks .	The Unsupervised machine learning models in which Autoencoders Perform well with F1 score of 91%.The authors explaination workflow of the system is well structured.
4	2021[4]	Generating network intrusion detection dataset based on real and encrypted synthetic attack traffic	*Applied Sciences*, *11*(17)	This paper addresses the challenges of the benchmark datasets present in the opensource domain and provides a new dataset.	The author is well aware of the current trends and statistics in the domain and has develop the dataset for the same. However, The author provides key insights to how the developed the dataset is collected.

2.1.1 Problem Definition

We are attempting to tackle the problem of detecting aberrant HTTP traffic, which might be symptomatic of security risks or system faults. Anomaly detection is a useful technique for detecting odd patterns or behaviours in data that are difficult to discover manually. Organisations may better safeguard their systems and data from possible attacks and other security concerns by utilising an anomaly detection system that can process and analyse massive amounts of HTTP data in real-time. The objective is to create a system that can distinguish between regular and aberrant HTTP traffic, provide warnings when odd behaviour is discovered, and allow for rapid examination and reaction to any security concerns or system issues.

2.1.2 Problem Analysis And Challenges

There are various hurdles to developing an anomaly detection system for HTTP traffic, including:

- Data volume: HTTP traffic can create a big quantity of data, which might be difficult to analyse and analyse in real time. To identify abnormal behaviour, the system must be able to handle and analyse massive amounts of data effectively.
- Data velocity: HTTP traffic might be generated at a rapid pace, and the system must be able to handle and analyse data in a timely way to detect unusual behaviour.
- Anomaly definition: Determining what an abnormality in HTTP traffic is sometimes be difficult since typical traffic patterns might differ significantly based on the website or application being accessed. The system must be able to create a baseline of typical behaviour and adjust to various traffic patterns.
- Security and privacy: Since HTTP data may contain sensitive information, the system must be built to prevent unauthorised access to or publication of this data. Additionally, the system must abide by data privacy laws to prevent legal and ethical issues

In conclusion ,Developing an anomaly detection system for HTTP traffic involves designing a system that can efficiently process and analyze large volumes of data to detect anomalous behavior ,while addressing challenges such as data velocity, and privacy.

2.1.3 Solution

To address these challenges, the system can use machine learning algorithms, statistical models, and other techniques to analyze HTTP data and detect anomalies. The system can also incorporate feedback loops to learn from false positives and improve accuracy over time. Additionally, the system must be designed with security and privacy in mind, using encryption and access controls to protect sensitive data.

It can be implemented in following way:

1. **Data collection:** Gather HTTP traffic data from different sources such as network devices, servers, and logs.

2. **Data preprocessing:** Clean and transform the collected data to prepare it for analysis, including handling missing values, removing duplicates, and normalizing the data.

3. **Feature engineering**: Extract relevant features from the HTTP data, such as request/response times, user agents, IP addresses, and resource types.

4. **Baseline establishment:** Create a baseline of normal HTTP traffic behavior by analyzing historical data, using statistical methods or machine learning algorithms.

5. **Anomaly detection:** Apply anomaly detection techniques such as clustering, classification, or time-series analysis to detect deviations from the established baseline.

6. **Alerting**: When an anomaly is detected ,it is to be to notified to security teams or system administrators of potential security threats or system issues.

7. **Investigation and response:** Investigate the detected anomalies to determine the cause and take appropriate actions to mitigate potential risks.

8. **Continuous improvement:** Incorporate feedback loops to learn from false positives and improve the accuracy of the anomaly detection system over time.

9. **Security and privacy:** Ensure that the system is designed with security and privacy in mind, using encryption, access controls, and other measures to protect sensitive data.

By implementing this solution, It is possible to make adjustments and modifications to the system over time as we know the cyber security threats evolve with time and it become harder to detect to them .

2.2 Requirements

2.2.1 Functional Requirements

1.Ability to collect and work on real world network data.

2. Automatic extraction of relevant information from network data.

3. Identification of normal and anamolous data.

4. Identification of anomalous data traffic that is not present in dataset.

5.Modularity of the system for modification for future purposes.

6. Ability to export the data and insights in csv formats for future purposes.

2.2.2 Non- Functional Requirements

1. Security and privacy of the data collected and analysed.

2. Accuracy and reliability of the detection model.

3. Availability and reliability of the tool.

4. Ease of use for users of varying technical abilities.

5. Compliance with relevant data protection and privacy regulations.

6. Efficient and effective data processing and analysis.

7. Continuous updating and improvement of the tool to stay current with the latest trends and developments in the industry.

2.3 Data- Flow Diagram (DFD)

- It starts with a network traffic analysis tool such as wireshark.
- We use tool to generate a pcap file of http data.
- Then the pcap file is fed to the systems GUI.
- The system converts the given pcap file into flows.
- Then preprocessing is carried out on the given data.
- This data is stored in a csv file.
- On opting for detection the the detection model runs in the background.
- After the detection is complete the results are stored in separate data stores and displaye to the user.

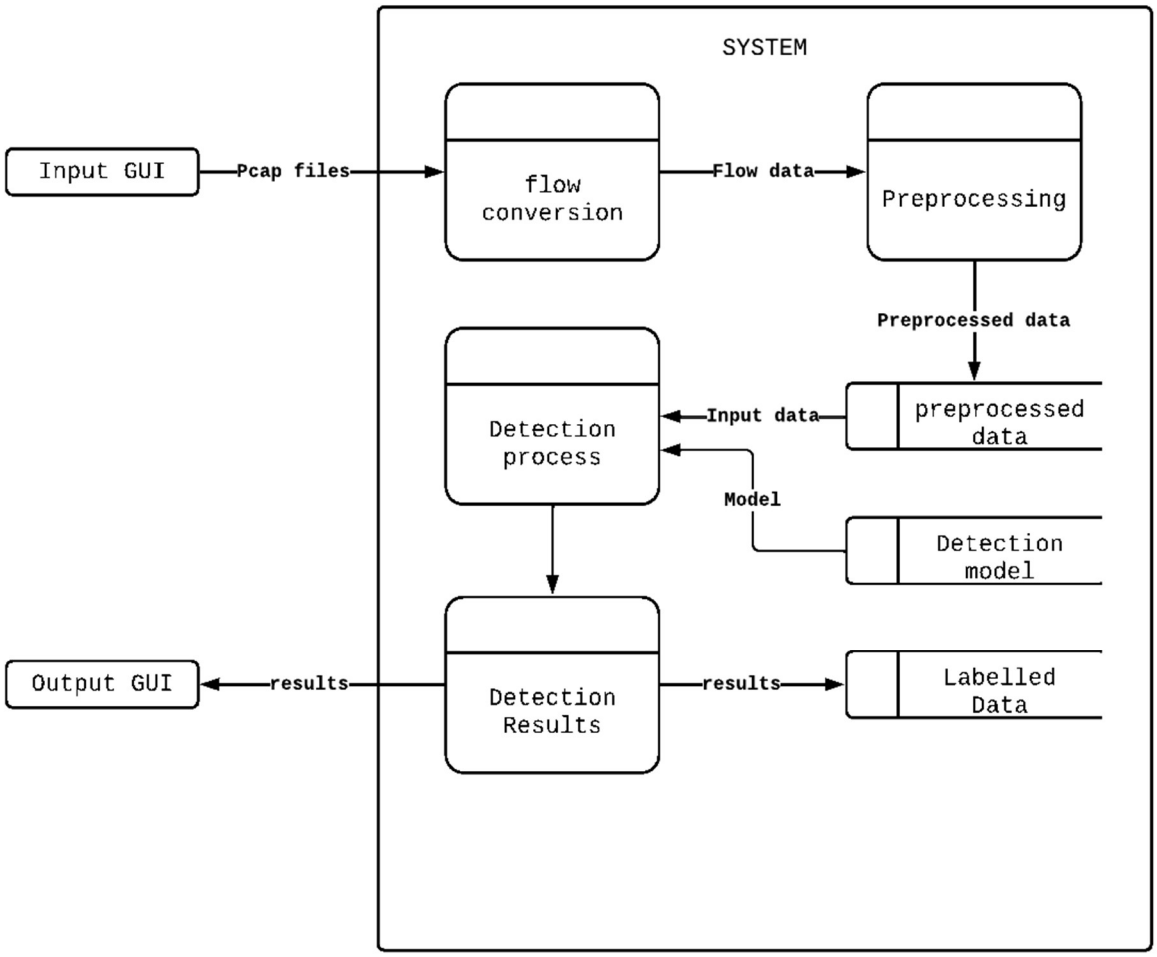


Figure 1. Data Flow Diagram(level 0)

Chapter 03 : Implementation

3.1 Dataset used

HIKARI – 2021

About

The HIKARI 2021 dataset is a publicly available cybersecurity dataset that contains comprehensive set of network traffic for cybersecurity research. The dataset is designed to support the research in intrusion detection, anomaly detection, and cybersecurity threat intelligence.

The dataset contains a diverse range of cyber attacks and threats, including malware infections, phishing attacks, brute force attacks, and denial of service attacks. The dataset is designed to support both supervised and unsupervised machine learning approaches to cybersecurity, and it includes labelled data for specific types of attacks and normal data.

It contains flow data of http network traffic. A flow refers to a sequence of data packets that share common attributes, such as source and destination IP addresses, source and destination port numbers, and protocol type . A flow can be thought of as a unidirectional conversation between two devices or applications over a network. Each flow is identified by a unique set of attributes, often referred to as a flow key, and can be used to monitor network traffic, track application performance, and identify security threats

Features

Sure, here are the features in the HIKARI2021 dataset and their explanations:

1. uid - Unique identifier for the flow.
2. originh - Source IP address of the flow.
3. originp - Source port number of the flow.
4. responh - Destination IP address of the flow.
5. responp - Destination port number of the flow.
6. flow_duration - Total duration of the flow in seconds.
7. fwd_pkts_tot - Total number of packets in the forward direction.
8. bwd_pkts_tot - Total number of packets in the backward direction.
9. fwd_data_pkts_tot – Total number of data packets in the forward direction.
10. bwd_data_pkts_tot - Total number of data packets in the backward direction.
11. fwd_pkts_per_sec - Number of packets per second in the forward direction.
12. bwd_pkts_per_sec - Number of packets per second in the backward direction.
13. flow_pkts_per_sec - Total number of packets per second in the flow.
14. down_up_ratio - Ratio of download speed to upload speed.
15. fwd_header_size_tot - Total size of headers in the forward direction.
16. fwd_header_size_min - Minimum header size in the forward direction.
17. fwd_header_size_max - Maximum header size in the forward direction.
18. bwd_header_size_tot - Total size of headers in the backward direction.

19. bwd_header_size_min - Minimum header size in the backward direction.
20. bwd_header_size_max - Maximum header size in the backward direction.
21. flow_FIN_flag_count - Number of FIN flags in the flow.
22. flow_SYN_flag_count - Number of SYN flags in the flow.
23. flow_RST_flag_count - Number of RST flags in the flow.
24. fwd_PSH_flag_count - Number of PSH flags in the forward direction.
25. bwd_PSH_flag_count - Number of PSH flags in the backward direction.
26. flow_ACK_flag_count - Number of ACK flags in the flow.
27. fwd_URG_flag_count - Number of URG flags in the forward direction.
28. bwd_URG_flag_count - Number of URG flags in the backward direction.
29. flow_CWR_flag_count - Number of CWR flags in the flow.
30. flow_ECE_flag_count - Number of ECE flags in the flow.
31. fwd_pkts_payload.min - Minimum payload size in the forward direction.
32. fwd_pkts_payload.max - Maximum payload size in the forward direction.
33. fwd_pkts_payload.tot - Total payload size in the forward direction.
34. fwd_pkts_payload.avg - Average payload size in the forward direction.
35. fwd_pkts_payload.std - Standard deviation of payload size in the forward direction.
36. bwd_pkts_payload.min - Minimum payload size in the backward direction.
37. bwd_pkts_payload.max - Maximum payload size in the backward direction.
38. bwd_pkts_payload.tot - Total payload size in the backward direction.
39. bwd_pkts_payload.avg - Average payload size in the backward direction.
40. bwd_pkts_payload.std - Standard deviation of payload size in the backward direction.
41. flow_pkts_payload.min - Minimum payload size in the flow.
42. flow_pkts_payload.max - Maximum payload size in the flow.
43. flow_pkts_payload.tot - Total payload size in the flow.
44. flow_pkts_payload.avg - Average payload size in the flow.
45. flow_pkts_payload.std - Standard deviation of payload size in the flow.
46. fwd_iat.min - Minimum inter-arrival time between packets in the forward direction.
47. fwd_pkts_payload.min- Minimum payload size of forward packets in the flow
48. fwd_pkts_payload.max- Maximum payload size of forward packets in the flow
49. fwd_pkts_payload.tot-Total payload size of forward packets in the flow
50. fwd_pkts_payload.avg-Average payload size of forward packets in the flow
51. fwd_pkts_payload.std- Standard deviation of payload size of forward packets in the flow
52. bwd_pkts_payload.min- Minimum payload size of backward packets in the flow
53. bwd_pkts_payload.max- Maximum payload size of backward packets in the flow
54. bwd_pkts_payload.tot- Total payload size of backward packets in the flow
55. bwd_pkts_payload.avg- Average payload size of backward packets in the flow
56. bwd_pkts_payload.std- Standard deviation of payload size of backward packets in the flow
57. flow_pkts_payload.min- Minimum payload size of packets in the flow
58. flow_pkts_payload.max- Maximum payload size of packets in the flow
59. flow_pkts_payload.tot- Total payload size of packets in the flow

60. flow_pkts_payload.avg- Average payload size of packets in the flow
61. flow_pkts_payload.std- Standard deviation of payload size of packets in the flow
62. fwd_iat.min- Minimum inter-arrival time of forward packets in the flow
63. fwd_iat.max- Maximum inter-arrival time of forward packets in the flow
64. fwd_iat.tot- Total inter-arrival time of forward packets in the flow
65. fwd_iat.avg- Average inter-arrival time of forward packets in the flow
66. fwd_iat.std- Standard deviation of inter-arrival time of forward packets in the flow
67. bwd_iat.min- Minimum inter-arrival time of backward packets in the flow
68. bwd_iat.max- Maximum inter-arrival time of backward packets in the flow
69. bwd_iat.tot- Total inter-arrival time of backward packets in the flow
70. bwd_iat.avg- Average inter-arrival time of backward packets in the flow
71. bwd_iat.std- Standard deviation of inter-arrival time of backward packets in the flow
72. flow_iat.min- Minimum inter-arrival time of all packets in the flow
73. flow_iat.max- Maximum inter-arrival time of all packets in the flow
74. flow_iat.tot- Total inter-arrival time of all packets in the flow
75. flow_iat.avg- Average inter-arrival time of all packets in the flow
76. flow_iat.std- Standard deviation of inter-arrival time of all packets in the flow
77. payload_bytes_per_second- Payload data rate in bytes per second
78. fwd_subflow_pkts- Number of packets in the forward subflow
79. bwd_subflow_pkts- Number of packets in the backward subflow
80. fwd_subflow_bytes- Number of bytes in the forward subflow
81. bwd_subflow_bytes- Number of bytes in the backward subflow
82. fwd_bulk_bytes- Number of bytes in the forward bulk transfer
83. bwd_bulk_bytes- Number of bytes in the backward bulk transfer
84. traffic_category- Categorical variable representing the traffic category of the network flow
85. Label- The label indicating the class of the network flow, i.e., whether it is a normal flow or an attack flow

The dataset is composition of numerical and categorical data.

Dimensions of the dataset:

Dats entries (555278 entries)

Data columns (86 columns)

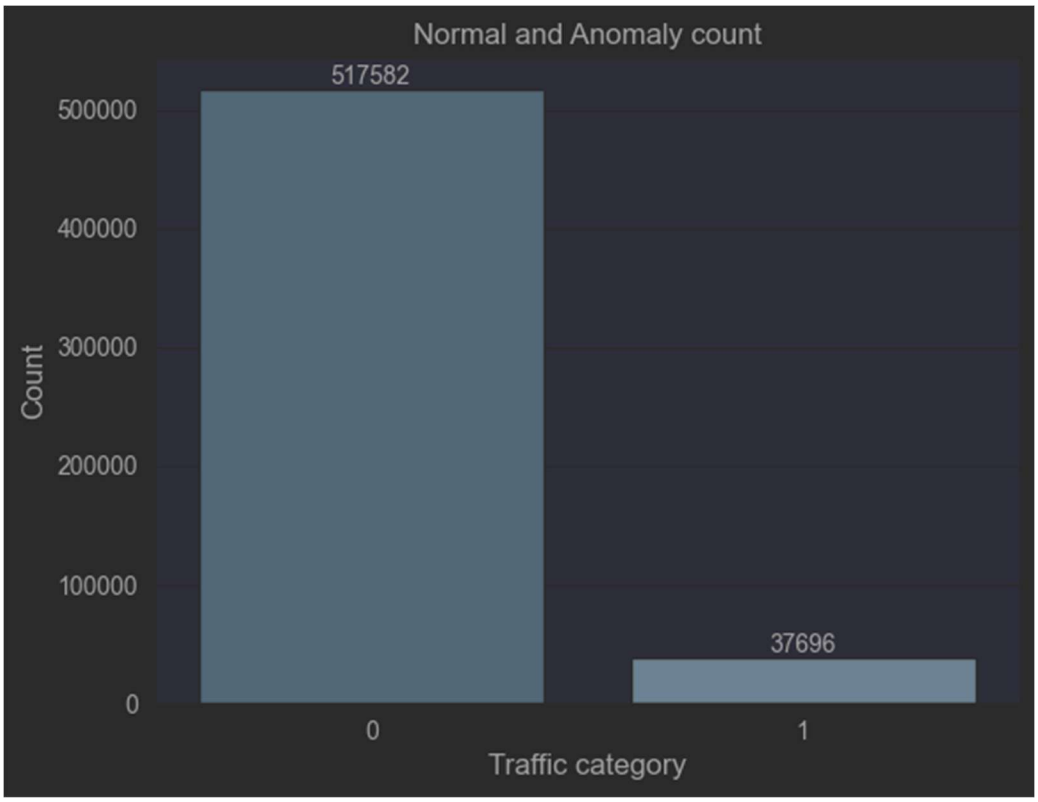


Figure 2. Normal and anomaly count in dataset

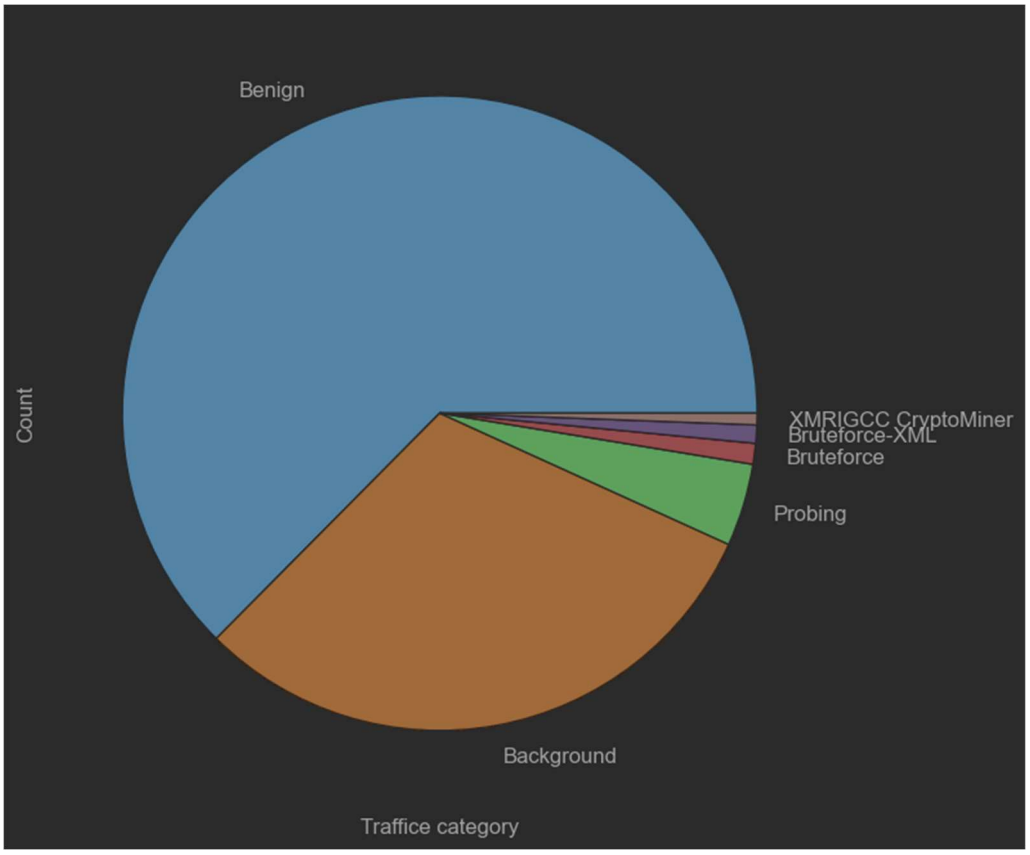


Figure 3. Traffic categories of dataset

3 Design of problem statement

3.3.1 Design of problem

The project is divided various modules as following-

- **Preprocessing of PCAP files**: The project includes preprocessing of the pcap files, such as feature extraction, normalization, and cleaning, to prepare them for classification.
- **Feature engineering:** The project should identify the relevant features from the preprocessed pcap files that can help in distinguishing between normal and anomalous traffic.
- **Machine learning models:** The project involves the development and evaluation of the machine learning models that can classify the traffic in the pcap files as normal or anomalous.
- **Performance evaluation:** The project includes performance evaluation and comparison of the developed models using relevant evaluation metrics such as accuracy, precision, recall, and F1-score.
- **Deployment:** The project involves the deployment of the developed machine learning models in a production environment for traffic classification of real world data.

3.3.2 Scope

- The project focuses on detection of anomalies that are mentioned in the used dataset or similar anomalies.
- The classification will be binary in nature i.e the anomalies will be labelled as 1 and normal data will be labelled as 0.
- The project seeks the detection of anomalies in real world network data.

3.3.3 End user

- This project is focused towards users who are active in the domain cyber security or are interested in the domain.
- Security researches who require system or methodologies of anomaly detection in HTTP data.

3.3.4 Expected outcomes

- Improved capability of detecting cyber attacks in a given HTTP network data and identifying potential threats.
- Enhanced awareness of changes in the network data of given application.

3.4 Algorithm/Pseudocode

For model generation-

1. Import Required libraries
2. Load dataset
3. Make list of categorical features, numerical features target features.
4. Extract important features from the given list using correlation
5. Select most important features into a input dataset
6. Apply preprocessing on the dataset
7. Split the dataset int training and testing sets.
8. Define the prediction models
9. Define threshold for the detection
10. Evaluate the results of the models
11. Select the best performing model
12. Save the model.

For application-

1. Import required libraries
2. Select a PCAP file from the storage location
3. Convert the PCAP file to flow data in CSV format
4. Preprocess the flow data and extract important features
5. Load the anomaly detection model from storage location
6. Input the preprocessed flow data into the loaded model to generate anomaly scores
7. Apply threshold to scores and generate labels (anomaly(1)) or normal(0))
8. Store the output scores into separate storage locations based on their labels
9. Load the output into a GUI window.

3.5 Flow chart
For model generation-

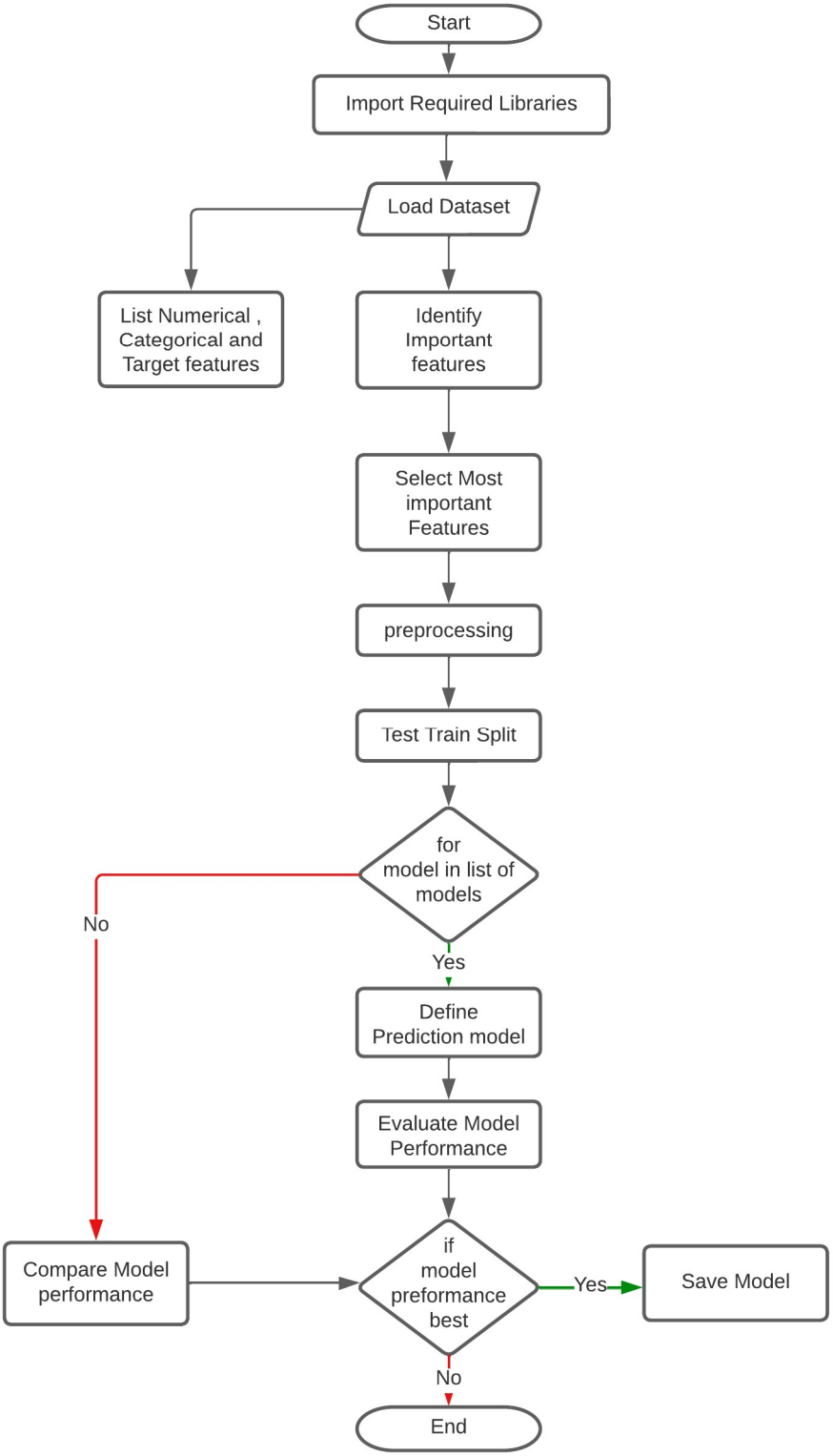


Figure 4. FlowChart for model generation

For application-

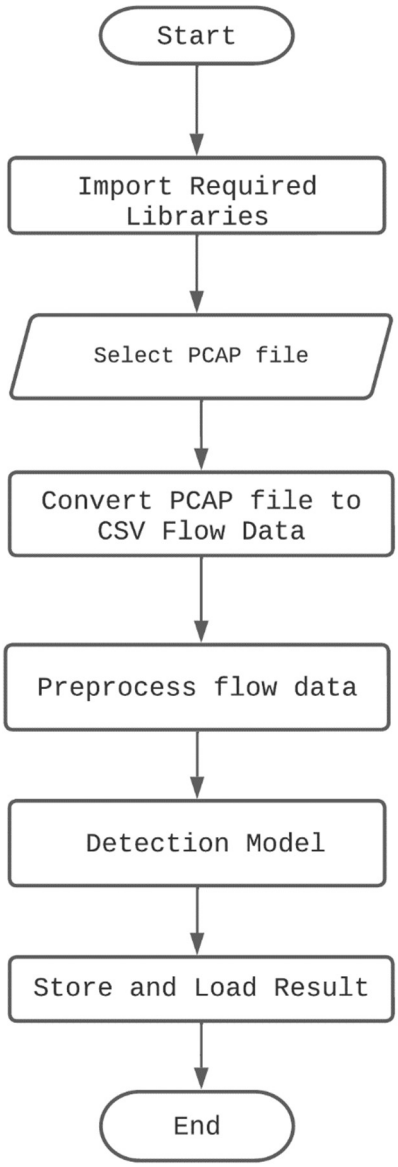


Figure 5. Flow Chart for application

3.6 Screenshots of the implementation

For Model Generation-

IMPORT STATEMENTS

```
#import libraies
import pandas as pd

import numpy as np
import tensorflow as tf

from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, f1_score, recall_score, accuracy_score,precision_score
```

```
#Load Dataset

df=pd.read_csv('Dataset/ALLFLOWMETER_HIKARI2021.csv')
```

FEATURE CATEGORIZATION

```
feature_columns = [
    col for col in df.columns
        if col not in id_columns + target_columns
]

numerical_columns = [
    col for col in feature_columns
        if df[col].dtype in [int, np.int64, float, np.float64]
           #Oblect type columns are not considered
]
categorical_columns = [
    col for col in feature_columns
        if col not in numerical_columns
]
print(len(numerical_columns), len(categorical_columns))
```

```
# Feature Extraction

x=df[feature_columns]
Y = df[target_columns[1]]
import seaborn as sns
import matplotlib.pyplot as plt
def find_corr_with_attr(df, attr):
    return df.corrwith(df[attr]).sort_values(ascending=False)

# Find correlation between target variable and all other attributes
corr_with_target = find_corr_with_attr(df, 'Label')

# Create a DataFrame with correlation coefficients of top 10 attributes
corr_df = pd.DataFrame(corr_with_target[:11], columns=['corr_coeff'])

# Create heatmap of correlation coefficients
sns.heatmap(corr_df, cmap='coolwarm', annot=True, vmin=-1, vmax=1, center=0, square=True, cbar=False)
plt.title('Correlation with Target Variable (Top 10 Attributes)')
plt.show()
```

```
# Feature selection

impFeatures=['bwd_init_window_size','fwd_pkts_payload.max','bwd_iat.min','fwd_iat.min',
             'fwd_subflow_bytes','fwd_pkts_payload.tot','fwd_pkts_payload.min','bwd_bulk_rate',
             'flow_pkts_payload.avg','down_up_ratio']
X=df[impFeatures]
y = df[target_columns[1]]
```

```
# preprocessing and train test split

from sklearn.preprocessing import MinMaxScaler

# Normalize the input features
scaler = MinMaxScaler()
X = scaler.fit_transform(X)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

ARTIFICIAL NEURAL NETWORKS

```
# Define the model
model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(18, activation='relu', input_shape=(10,)),
  tf.keras.layers.Dense(14, activation='relu'),
  tf.keras.layers.Dense(10, activation='relu'),
  tf.keras.layers.Dense(6, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=6, validation_data=(X_test, y_test))

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
model.save('model.h5')
model.summary()
#result(X_train,X_test)
print('Test Accuracy:', test_acc)
```

```python
# Loop through threshold values
thresholds = np.arange(0.05, 1, 0.05)
f1_scores = []
for threshold in thresholds:
    # Convert probabilities to binary predictions using the threshold
    y_pred_binary = (y_pred > threshold).astype('int32')
    # Calculate F1 score
    f1 = f1_score(y_test, y_pred_binary)
    # Add F1 score to list of scores
    f1_scores.append(f1)

# Find the threshold that gives the highest F1 score
best_idx = np.argmax(f1_scores)
best_threshold = thresholds[best_idx]

# Print best threshold and corresponding F1 score
print('Best threshold:', best_threshold)
```

```python
# Load the saved model
loaded_model = tf.keras.models.load_model('model.h5')

# Make predictions on the test set
y_pred = loaded_model.predict(X_test)

# Convert probabilities to binary predictions
y_pred_binary = (y_pred > 0.25).astype('int32')
print(y_pred,y_pred_binary)
# Calculate evaluation metrics
test_acc = accuracy_score(y_test, y_pred_binary)
f1 = f1_score(y_test, y_pred_binary)
recall = recall_score(y_test, y_pred_binary)
pi=precision_score(y_test, y_pred_binary)
# Print evaluation metrics
print('Test Accuracy:', test_acc)
print('F1 Score:', f1)
print('Recall:', recall)
print('precision:', pi)
```

```
# stacked auto encode implementation 3 levels

from keras import Sequential
from keras.layers import Dense
# Build the first autoencoder
model1 = Sequential()
model1.add(Dense(10, activation='relu', input_shape=(X_train.shape[1],)))
model1.add(Dense(4, activation='relu'))
model1.add(Dense(10, activation='relu'))
model1.add(Dense(X_train.shape[1], activation='sigmoid'))
model1.compile(optimizer='adam', loss='mean_squared_error')

# Train the first autoencoder
model1.fit(X_train, X_train, epochs=10, batch_size=32, validation_data=(X_test, X_test))
```

```
# Train the first autoencoder
model1.fit(X_train, X_train, epochs=10, batch_size=32, validation_data=(X_test, X_test))

# Extract the output of the first autoencoder
encoder1 = Sequential()
encoder1.add(model1.layers[0])
encoder1.add(model1.layers[1])
encoder1.add(model1.layers[2])
encoder1.compile(optimizer='adam', loss='mean_squared_error')
encoded_X_train = encoder1.predict(X_train)
encoded_X_test = encoder1.predict(X_test)

# Build the second autoencoder
model2 = Sequential()
model2.add(Dense(16, activation='relu', input_shape=(encoder1.output_shape[1],)))
model2.add(Dense(8, activation='relu'))
model2.add(Dense(16, activation='relu'))
model2.add(Dense(encoder1.output_shape[1], activation='sigmoid'))
model2.compile(optimizer='adam', loss='mean_squared_error')

# Train the second autoencoder
model2.fit(encoded_X_train, encoded_X_train, epochs=10, batch_size=32, validation_data=(encoded_X_test, encoded_X_test))

# Extract the output of the second autoencoder
encoder2 = Sequential()
encoder2.add(model2.layers[0])
encoder2.add(model2.layers[1])
encoder2.compile(optimizer='adam', loss='mean_squared_error')
encoded_X_train_final = encoder2.predict(encoded_X_train)
```

```
encoded_X_test_final = encoder2.predict(encoded_X_test)

# Add classification layer and fine-tune the model
model = Sequential()
model.add(encoder2)
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the performance
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Make predictions on the test set
y_pred = model.predict(X_test)
y_pred = [1 if p>=0.25 else 0 for p in y_pred]

# Calculate evaluation metrics
acc = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print evaluation metrics
print('Accuracy:', acc)
print('Precision:', precision)
print('Recall:', recall)
print('F1 Score:', f1)
```

```
# Random forest classification

# Import necessary libraries
from sklearn.ensemble import RandomForestClassifier
# Train a random forest classifier
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the performance of the classifier
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print evaluation metrics
print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
print('F1 Score:', f1)
```

For application-

```
import customtkinter

from browse_window import BrowseWindow


class App(customtkinter.CTk):
    def __init__(self):
        super().__init__()
        self.geometry("600x320")
        self.title("Anomaly Detection System")

        # Add widgets to app
        self.label = customtkinter.CTkLabel(self, text="Anomaly Detection System", text_color="light blue",
                                            anchor="n", font=("Helvetica", 14, "bold"))
        self.label.grid(row=0, column=6, padx=20, pady=10)
        self.textbox = customtkinter.CTkTextbox(master=self, width=600, corner_radius=0)
        self.textbox.grid(row=2, column=6, pady=10)
        self.textbox.insert("0.0", "Anomaly Detection System for HTTP Flow Data\n"
                                   " customization and integration with existing security infrastructure, making it \n"
                                   "scalable and adaptable to different network environments and security requirements.\n"
                                   "\nIt provides a valuable tool for enhancing network security posture and mitigating \n"
                                   "potential security risks by proactively identifying and responding to anomalous\n"
                                   " behaviors in HTTP flow data.\nAuthor: [Aatish Sharma , Animesh]\nRoll Numbers: [201536 ,201567]\n")
        self.button = customtkinter.CTkButton(self, text="OK", command=self.button_click, hover_color="crimson")
        self.button.grid(row=6, column=6, padx=20, pady=10)

    def button_click(self):
        app.destroy()
        browse_window = BrowseWindow()
        browse_window.mainloop()


# Create app window
app = App()
try:
    # Run the main event loop

    app.mainloop()

except KeyboardInterrupt:
    # Handle keyboard interrupt
    app.destroy()
```

```
import customtkinter
from tkinter import END
from cmand_interface import browse_input_file
from post_prediction.processing_window import ResultWindow
class BrowseWindow(customtkinter.CTk):
    def __init__(self):
        super().__init__()
        self.title("Anomaly Detection")

        # Create a label
        label = customtkinter.CTkLabel(self, text="Select File .pcap file for  analysis:",text_color="light blue",font=("Helvetica", 14, "normal"))
        label.grid(row=0, column=0, sticky="w", padx=20, pady=10)

        # Create a button to open file dialog
        button = customtkinter.CTkButton(self, text="Browse", command=self.on_browse_button_click, hover_color="crimson")
        button.grid(row=1, column=0, sticky="w", padx=20)

        # Report window
        label1 = customtkinter.CTkLabel(self, text="Report:", text_color="light blue", anchor="w",font=("Helvetica", 14,"normal"))
        label1.grid(row=2, column=0, sticky="w", padx=20, pady=10)
        self.textbox1 = customtkinter.CTkTextbox(self, width=600, corner_radius=0)
        self.textbox1.configure(state="disabled")
        self.textbox1.grid(row=3, column=0, sticky="w", padx=20)
        # Detect button
        button_detect = customtkinter.CTkButton(self, text="Detect", command=self.on_detect_button_click, hover_color="green")
        button_detect.grid(row=4, column=0, sticky="e", padx=20,pady=20)

        # Clear button
        button_clear = customtkinter.CTkButton(self, text="Clear", command=self.clear_report, hover_color="red")
        button_clear.grid(row=4, column=0, sticky="w", padx=20,pady=20)
```

```
def on_browse_button_click(self):
    # Function to handle Browse button click event
    file_path = browse_input_file()
    # Update the report text box with the selected file path
    self.textbox1.configure(state='normal')
    self.textbox1.insert(END, file_path+'has been added and processed\n')
    self.textbox1.configure(state='disabled')

def on_detect_button_click(self):
    # Function to handle Detect button click event
    # Add your detection logic here
    result= ResultWindow()
    self.destroy()
    result.mainloop()

   # result.update()

def clear_report(self):
    self.textbox1.configure(state='normal')
    self.textbox1.delete(1.0, END)
    self.textbox1.configure(state='disabled')
```

```
def on_browse_button_click(self):
    # Function to handle Browse button click event
    file_path = browse_input_file()
    # Update the report text box with the selected file path
    self.textbox1.configure(state='normal')
    self.textbox1.insert(END, file_path+'has been added and processed\n')
    self.textbox1.configure(state='disabled')

def on_detect_button_click(self):
    # Function to handle Detect button click event
    # Add your detection logic here
    result= ResultWindow()
    self.destroy()
    result.mainloop()

   # result.update()
```

```python
import os
from tkinter import filedialog


# Function to browse and select input file
def browse_input_file():
    file_path = filedialog.askopenfilename()
    if file_path:
        # Call function to process input file
        process_files(file_path)
        return str(file_path)
    else:
        return "No file "

# Function to process input file
def process_files(file_path):
    # Change directory to "j"
    os.chdir("J:")

    # Construct the path to the batch file
    batch_file_path = os.path.join("bin", "cfm")

    # Construct the path to the CSV output file

    csv_file_path = os.path.join("C:Users/aatis/PycharmProjects/pythonProject1/Dataset/input/data")

    # Run the batch file with the input file as input and the csv file as output
    command = f'{batch_file_path} {file_path} {csv_file_path}'
    os.system(command)
    print("Files processed successfully.")
```

```python
import pandas as pd
import os


folder_path = os.path.join("C:"،"Users"،"aatis"،"PycharmProjects"،"pythonProject1", "Dataset", "input", "data")
all_data = pd.DataFrame()  # Create an empty DataFrame to store concatenated data

for filename in os.listdir("C:/Users/aatis/PycharmProjects/pythonProject1/Dataset/input/data"):
    if filename.endswith(".csv"):
        file_path = os.path.join("C:/Users/aatis/PycharmProjects/pythonProject1/Dataset/input/data", filename)
        # Create the full file path
        data = pd.read_csv(file_path)  # Read CSV file
        all_data = pd.concat([all_data, data], ignore_index=True)  # Concatenate data to the main DataFrame
        #os.remove(file_path)

folder_path_processed=os.path.join("C:/Users/aatis/PycharmProjects/pythonProject1/Dataset/preprocessed_input")
# Save preprocessed data to a new CSV file
preprocessed_file_path = os.path.join(folder_path_processed, "preprocessed_data.csv")
all_data.to_csv(preprocessed_file_path, index=False)
```

```python
import pandas as pd


def preprocess():
    # collective dataset that stores the data in a single file format.
    df = pd.read_csv("C:/Users/aatis/PycharmProjects/pythonProject1/Dataset/preprocessed_input/preprocessed_data.csv")
    # preprocessing input record
    impFeatures = ['Init Bwd Win Byts', 'Fwd Pkt Len Max', 'Bwd IAT Min', 'Fwd IAT Min', 'Subflow Fwd Byts',
                   'TotLen Fwd Pkts', 'Fwd Pkt Len Min', 'Bwd Blk Rate Avg', 'Pkt Size Avg',
                   'Down/Up Ratio']
    X = df[impFeatures]
    X=X.dropna()
    from sklearn.preprocessing import MinMaxScaler

    # Normalize the input features
    scaler = MinMaxScaler()
    X = X.values
    X = scaler.fit_transform(X)

    return X
```

```python
import pandas as pd
import numpy as np
import tensorflow as tf
from pre_prediction import preprocess_input
model = tf.keras.models.load_model('C:/Users/aatis/PycharmProjects/pythonProject1/model_interfaces/model.h5')
def csv_to_string(file_path, delimiter=',', encoding='utf-8'):
    try:
        # Load the CSV file into a pandas data frame
        df = pd.read_csv(file_path, delimiter=delimiter, encoding=encoding)

        # Convert the data frame to a string and return it
        return df.to_string(index=False)
    except Exception as e:
        print('Error:', str(e))
        return None
```

```python
def load():
    # Load new data from a CSV file
    new_data =preprocess_input.preprocess()
    # Convert the data type of the NumPy array to a supported type
    new_data = new_data.astype(np.float32)
    # Make predictions on the new data
    y_pred = model.predict(new_data)
    # Convert the NumPy array to a Pandas DataFrame
    new_data = pd.DataFrame(new_data)
    # Convert probabilities to binary predictions using a threshold
    y_pred_binary = (y_pred > 0.2).astype('int32')
    # Create a new column in the data frame with predicted labels
    new_data['predicted_label'] = y_pred_binary

    # Create two separate data frames for positive and negative predictions
    positive_df = new_data[new_data['predicted_label'] == 1]
    negative_df = new_data[new_data['predicted_label'] == 0]

    # Save the two data frames to separate CSV files
    positive_df.to_csv('positive_predictions.csv', index=False)
    negative_df.to_csv('negative_predictions.csv', index=False)

    # Convert the positive and negative CSV files to strings
    positive_csv_string = csv_to_string('positive_predictions.csv')
    negative_csv_string = csv_to_string('negative_predictions.csv')

    return positive_csv_string,negative_csv_string
```

Chapter 04 : Results

4.1 Results

On the basis of analysis of features and there importance we selected 10 of the most important features that were not derived from any existing features in he dataset and trained three different models on it.

Those models are:

- Artificial Neural Networks
- Random Forest Classifiers
- Stacked Auto Encoders.

The performance of these models are as follows:

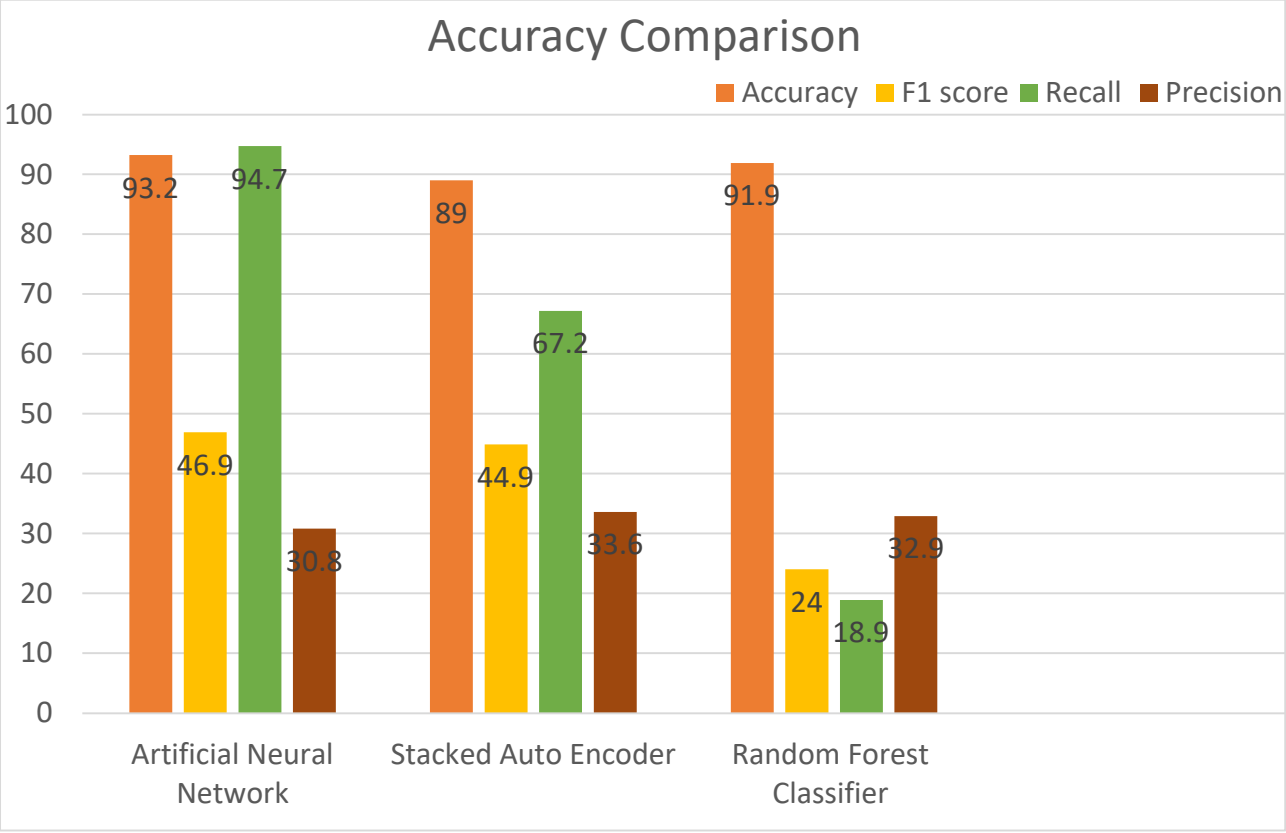


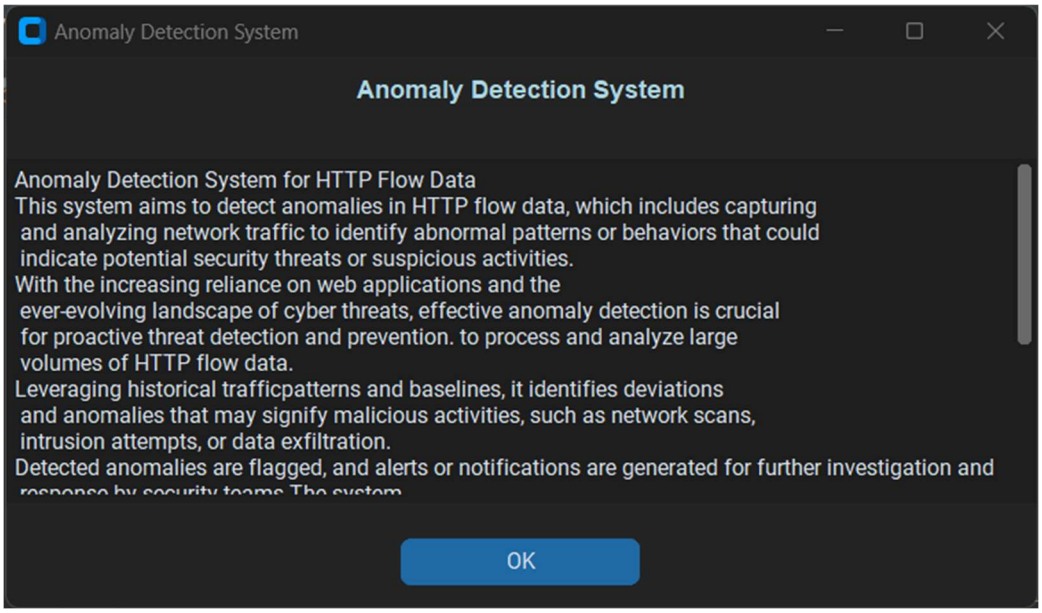
Figure 6. Accuracy of the models

- The Artificial Neural Network is fast and has a moderate accuracy in comparison with other techniques.
- Random forest Classifier is faster than other techniques but accuracy is low in comparison.
- The autoencoders is less accurate and is not practical in terms of detection time.

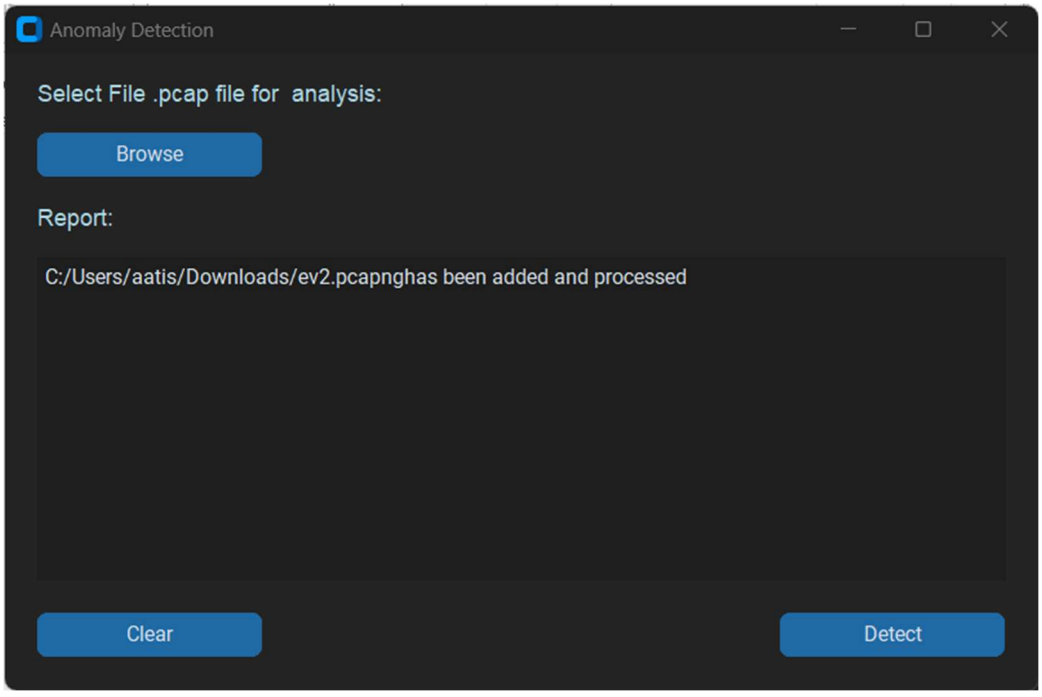
From the above evaluation it is clear that ANN out performs all other techniques. So, We selected Artificial Neural Networks for Deployment purposes.

For Deployment we developed a system that takes pcap as input and returns predictions as an output on that data which can be used using the GUI.

Welcome window-



File selection Window-



Result window-

4.2 Application of the project :

- This project can be use a network analysis tool for HTTP data.
- Due its modular design it can also be used for testing different models and techniques.
- It can also be used for research purposes as the predicted data is saved for the future use. User can generate his own database by performing attacks from a controlled and known device.

4.3 Limitations of the project

- The Dataset used contains only 4 types which results in a model that is capable of detecting limited types of attacks.
- The detection of anomalies in Network data is a memory and processing extensive task and requires high memory and compute capacity.
- Due to limited compute resources the project is only capable of analysing saved network data.
- The Accuracy of the detection is moderate and result in high false positive detections.

4.4 Future work

This project expanded our knowledge of cyber security and machine learning further than before. In the future we plan to tackle limitations and make this project into real time application that is capable of detecting anomalies in live network traffic. The detection models used in this project are made with the help of opensource libraries, In future we would make these models from scratch without the use of any external libraries. The detection techniques used in this project are simple and more complex , effective techniques are needed to be applied and tested for better results.

Abbreviations

DNNs: Deep neural networks; FSA: Finite-state automaton; HMM: Hidden Markov models; JVM: Java virtual machine; LSTM: Long short-term memory; PCA: Principle component analysis; RSMT: Robust software modeling tool; SVM: Support vector machine; XSS: Cross site scripting ;IF: Isolation Forest; SAE: Stacked Auto Encoder ; DBN: Deep Belief Network.

References

[1] Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly Detection : A Survey. *ACM Computing Surveys*, 1-13.

[2] Moradi Vartouni, A., Teshnehlab, M., & Sedighian Kashi, S. (2019). Leveraging deep neural networks for anomaly-based web application firewall. *IET Information Security*, *13*(4), 352361.

[3] Pan, Y., Sun, F., Teng, Z., White, J., Schmidt, D. C., Staples, J., & Krause, L. (2019). Detecting web attacks with end-to-end deep learning. *Journal of Internet Services and Applications*, *10*(1), 1-22.

[4] Ferriyan, A., Thamrin, A. H., Takeda, K., & Murai, J. (2021). Generating network intrusion detection dataset based on real and encrypted synthetic attack traffic. *Applied Sciences*, *11*(17), 7868.

[5] Ghazal, S. F., & Mjlae, S. A. (2022). Cybersecurity in Deep Learning Techniques: Detecting Network Attacks. *International Journal of Advanced Computer Science and Applications*, *13*(11).