**Industry Oriented Mini Project Report**

**on**

# ENHANCING INTRUSION DETECTION SYSTEM USING XDP WITH EBPF

Submitted in partial fulfillment of the requirements
for the award of degree of

BACHELOR OF TECHNOLOGY

in

Information Technology

by

**K. Geethika Reddy (20WH1A1270)**

**G. Krishna Prathibha (20WH1A12B0)**

**G. Sneha (20WH1A12B5)**

Under the esteemed guidance of

**Mr. N. Anand**

Assistant Professor


Department of Information Technology

BVRIT HYDERABAD College of Engineering for Women

Rajiv Gandhi Nagar, Nizampet Road, Bachupally, Hyd–500090

(Affiliated to Jawaharlal Nehru Technological University, Hyderabad)

(NAAC 'A' Grade & NBA Accredited- ECE, EEE, CSE & IT)

Dec 19, 2023


# BVRIT HYDERABAD

## College of Engineering for Women

**Rajiv Gandhi Nagar, Nizampet Road, Bachupally, Hyderabad – 500090**
**(Affiliated to Jawaharlal Nehru Technological University Hyderabad)**
**(NAAC 'A' Grade & NBA Accredited- ECE, EEE, CSE & IT)**

## CERTIFICATE

This is to certify that the Project report on **" Enhancing Intrusion Detection System using XDP with eBPF "** is a bonafide work carried out by **K Geethika Reddy (20WH1A1270), G Krishna Prathibha(20WH1A12B0) and G Sneha(20WH1A12B5)** in the partial fulfillment for the award of B.Tech degree in **Information Technology , BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad** affiliated to Jawaharlal Nehru Technological University, Hyderabad, under my guidance and supervision. The results embodied in the project work have not been submitted to any other university or institute for the award of any degree or diploma.

**Internal Guide**
**Mr. N. Anand**
**Assistant Professor**
**Department of IT**

**Head of the Department**
**Dr. Aruna Rao S L**
**Professor & HoD**
**Department of IT**

**External Examiner**

# DECLARATION

We hereby declare that the work presented in this project entitled **"Enhancing Intrusion Detection System using XDP with eBPF"** submitted towards completion of in IV year I sem of B.Tech IT at "BVRIT HYDERABAD College of Engineering for Women",Hyderabd is an authentic record of our original work carried out under the esteemed guidance of **Mr. N. Anand, Assistant Professor**, Department of Information Technology.

K. Geethika Reddy (20WH1A1270)

G. Krishna Prathibha (20WH1A12B0)

G. Sneha (20WH1A12B5)

*This project report is dedicated to my beloved Family members and supervisor for their limitless support and encouragement and to you as a reader*

# ACKNOWLEDGMENTS

We would like to express our profound gratitude and thanks to **Dr. K.V.N. Sunitha, Principal, BVRIT HYDERABAD College of Engineering for Women** for providing the working facilities in the college.

Our sincere thanks and gratitude to **Dr. Aruna Rao S L, Professor & Head, Department of IT, BVRIT HYDERABAD College of Engineering for Women** for all the timely support, constant guidance and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, **Mr. N. Anand, Assistant Professor Department of IT, BVRIT HYDERABAD College of Engineering for Women** for her constant guidance, encouragement and moral support throughout the project.

Finally, we would also like to thank our Project Coordinators **Mr. Ch Anil Kumar, Assistant Professor and Mr. N Anand, Assistant Professor**, all the faculty and staff of Department of IT who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

K. Geethika Reddy (20WH1A1270)

G. Krishna Prathibha (20WH1A12B0)

G. Sneha (20WH1A12B5)

# ABSTRACT

Intrusion Detection Systems (IDS) play critical role in safeguarding network infrastructure from unauthorized access and malicious activities. This paper explores the integration of XDP (eXpress Data Path) technology with eBPF (extended Berkeley Packet Filter) to enhance the performance and capabilities of an IDS in detecting DOS(Denial of Service), DDOS(Distributed Denial of Service) attacks at Kernal space. eBPF enables custom code execution within the Linux kernel, while XDP provides a high-performance data path for packet processing. XDP programs are written in eBPF bytecode, and are attached to network devices. By intercepting incoming network packets at the NIC, XDP enables rapid processing before reaching the kernel network stack and act on the packet directly on the NIC. This combined XDP and eBPF approach represents a potent advancement in network security, providing a robust toolset for defending against a wide range of modern cyber threats, including DoS and DDoS attacks. By leveraging these technologies, we aim to achieve faster and more efficient packet analysis, enabling the IDS to respond to threats in real time.

***Keywords***: IDS, XDP, eBPF, DOS, DDOS

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

An Intrusion Detection System (IDS) is a vital cybersecurity tool that monitors network or system activities to identify potential security threats. It is a network function that examines packets for detecting network attacks. It checks the content of headers and payload of packets to detect intrusions from the network. IDS is an essential function for network security as it helps to identify and respond to potential security threats in real-time. It can detect various types of attacks, including malware, viruses, and unauthorized access attempts, and alert network administrators to take appropriate actions to prevent further damage.

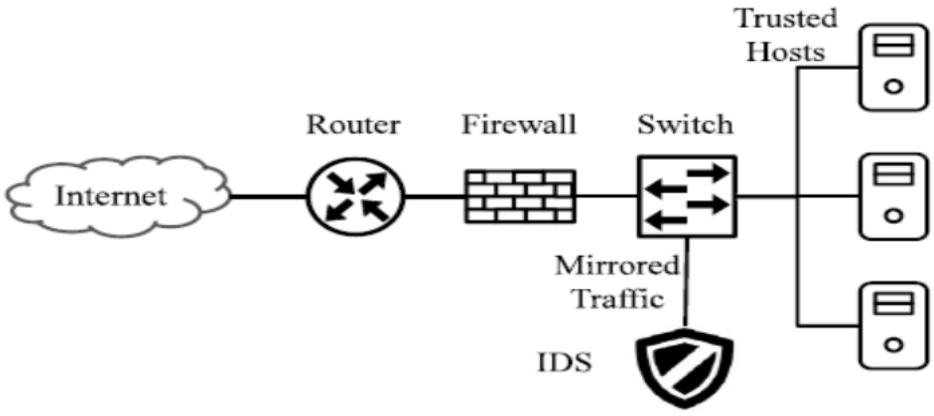


Figure 1.1: A typical usage of IDS

## 1.2 Objective

The primary objective of ENHANCING INTRUSION DETECTION SYSTEM USING XDP WITH eBPF is to comprehensively explore the realm of fast packet processing through the innovative technologies of eBPF (extended Berkeley Packet Filter) and XDP (eXpress Data Path). Beginning with a succinct introduction, we aim to provide readers with a clear understanding of the origins, evolution, and integration of eBPF and XDP into the Linux kernel. Moving beyond the introductory phase, we delve into the fundamental concepts and architecture that underpin these technologies, elucidating the virtual machine model and the intricacies of bytecode execution within the kernel.A central focus of our exploration is the practical application of eBPF and XDP in the context of packet processing. We present a series of code examples to illustrate the seamless integration of eBPF programs for tasks such as packet filtering and modification. By doing so, we aim to empower readers with a hands-on understanding of how these technologies can be leveraged to achieve fast and efficient packet processing.Furthermore, our project aims to shed light on the substantial performance benefits offered by eBPF and XDP in comparison to traditional packet processing methods. Through the presentation of benchmark results and real-world use cases, we highlight instances where the implementation of eBPF and XDP has yielded remarkable speed improvements. However, we also acknowledge and discuss the challenges associated with these technologies, providing a balanced perspective on their strengths and limitations.

## 1.3 Problem Statement

The problem at hand involves the efficiency of current attack detection methods, such as IDS and IPS, in identifying application-level attacks like HTTP and DHCP within the network layer. While these methods are capable of detection, the process of packet filtering at the network layer poses challenges, leading to potential loss of data or valuable information. Factors contributing to this loss include DOS attacks, resource limitations, and inadequate throughput. To address these issues and enhance performance, the study aims to leverage machine learning algorithms for the classification of normal and attack data. The central question guiding the investigation is, "At what level within the network architecture should attacks be filtered to optimize detection accuracy and mitigate data loss, thereby improving overall system performance compared to existing methods?"

The aim of the project is to enhance the efficiency of attack detection and mitigation methods within the network layer, specifically focusing on application-level attacks such as HTTP and DHCP. The project seeks to address challenges associated with potential data loss during packet filtering, attributed to issues like DOS attacks, resource limitations, and inadequate throughput. By incorporating machine learning algorithms, the objective is to classify normal and attack data more accurately, ultimately improving the overall performance compared to existing methods. The project aims to identify the optimal level within the network architecture for filtering attacks, striking a balance between heightened security measures and the preservation of critical information while maintaining optimal system throughput.

# Chapter 2

# Literature Survey

## 2.1 Related Work

Anand, Nemalikanti, Saifulla, and Aakula[1], introduces eBPF as a bytecode-based virtual machine capable of executing programs without necessitating modifications to the kernel source code. The authors advocate for leveraging eBPF to capture network traffic and extract features for training machine learning models, particularly focusing on intrusion detection for DoS and DDoS attacks. The study evaluates the performance of four machine learning algorithms (DT, RF, SVM, TwinSVM) in the context.

Vieira, Marcos AM, Castanho, Matheus S, and Pac [2], the paper delivers an in-depth exploration of eBPF (extended Berkeley Packet Filter) technology, enabling user-defined program execution within Unix-based operating system kernels. It traces the evolution of eBPF from its precursor, Berkeley Packet Filter (BPF), detailing its architecture and functionalities, encompassing packet filtering, tracing, and dynamic tracing. The paper scrutinizes eBPF's limitations and proposes strategies for overcoming.

Wang, Shie-Yuan and Chang, Jen-Chieh[3], paper presents a design and implementation of an intrusion detection system (IDS) using eBPF in the Linux kernel. The proposed IDS consists of two parts: an eBPF program running in the kernel to pre-check and pre-drop packets, and a program running in user space to further examine the packets left by the first part. The authors demonstrate that their IDS outperforms Snort, a widely used IDS, by a factor of 3 under many tested conditions. The paper provides a comprehensive comparison of related work, highlighting the limitations of existing IDS solutions. The authors argue that their proposed IDS is more efficient

and effective than existing solutions, due to its use of eBPF and XDP.

Høiland-Jørgensen, Toke and Brouer, Jesper Dangaard and Borkmann, Daniel and Fastabend[4], introduces the eXpress Data Path (XDP), a new approach to programmable packet processing that provides a safe execution environment for custom packet processing applications executed in device driver context. XDP is part of the mainline Linux kernel and provides a fully integrated solution working in concert with the kernel's networking stack. The paper outlines related work, presents the design of the XDP system, evaluates its raw packet processing performance, and supplements this with examples of real-world use cases that can be implemented with XDP. Overall, the paper demonstrates that XDP enables high-performance packet processing while ensuring the safety and integrity of the rest of the system, and is flexible and can be used for a variety of use cases.

Aljanabi, Mohammad and Ismail, Mohd Arfian and Ali, Ahmed Hussein[5], article explains that intrusion detection systems (IDSs) face challenges in maintaining an up-to-date profile as new protocols evolve over time. This is particularly true for ID systems that rely on stateful protocol analysis, which exhibit different detection performances based on the level of their profile definition. Essentially, as new protocols are developed and used, IDSs need to be able to recognize and analyze them in order to detect potential intrusions. However, this can be difficult to do in real-time, and keeping profiles up-to-date can be a challenge.

Bachl, Maximilian and Fabini, Joachim and Zseby, Tanja[6], explains the creation of a machine learning-based flow-based intrusion detection system (IDS) with eBPF technology is covered in the file. The system considers the complete preceding context of the network flow while utilizing a decision tree to assess if a given packet is malicious or not. When compared to the same solution implemented as a userspace program, the performance boost is greater than 20%. The file also makes recommendations for future research to determine if using deep neural networks or random forests in eBPF can also result in a performance boost.

The Wieren, HD[7], study is centered on how to protect against DDoS attacks using eBPF and XDP. The study offers a thorough analysis of the situation of DDoS attacks today and the shortcomings of the mitigation strategies currently in use. After that, the study presents eBPF and XDP and describes how these technologies can get around some of the drawbacks of existing techniques. The paper also explains how user-space collaboration with other

apps is made possible using eBPF and XDP, which may be implemented on a broad range of devices.In addition to offering fresh perspectives to the scientific community, the study can assist network operators in protecting their systems from DDoS attacks.

Panigrahi, Ranjit and Borah, Samarjeet[8], paper provides a detailed analysis of the CICIDS2017 dataset for designing Intrusion Detection Systems (IDS). The dataset contains five days of normal and attack traffic data from the Canadian Institute of Cybersecurity. The paper identifies and provides effective solutions to the shortcomings of the dataset, including high class imbalance problem, and relabels the dataset to reduce this issue. The analysis shows that the CICIDS2017 dataset is a valuable resource for developing and testing IDS models and algorithms. The paper provides a comprehensive description of the dataset and its characteristics, as well as outlining the limitations and challenges in using it for IDS design. The insights from this analysis can be applied in designing effective IDS that can detect and prevent cyber attacks.

Leblond, E and Manev, Peter[9], paper provides an overview of eBPF and XDP support in Suricata, a network threat detection engine. The paper explains how Suricata's flow bypass feature helps to limit the impact of "elephant flows" that can cause overflow in packet acquisition buffers. It also discusses the benefits of using eBPF and XDP support in Suricata, including improved performance and the ability to detect and prevent network threats more effectively. The paper concludes by briefly mentioning Stamus Networks' commercial solutions and their potential integration with Suricata for enhanced network security.

Packet, Berkeley[10], paper provides a comprehensive overview of XDP-Programmable Data Path in the Linux Kernel. The article begins by discussing the history of Berkeley Packet Filter (BPF) and its evolution into extended BPF (eBPF), which is a more powerful and flexible version of BPF. The article describes how eBPF works and its key features, including its ability to execute bytecode in the data path of the networking stack. The article also explains how eBPF can be used to create a fast and efficient firewall using Express Data Path (XDP). The article provides a step-by-step guide on how to implement a firewall using XDP and eBPF. The guide includes instructions on how to set up a Linux environment, how to write and compile an eBPF program, and how to load the program into the kernel using the XDP framework. The article also provides examples of eBPF programs that can be used for packet filtering and monitoring.

# Chapter 3

# System Design

## 3.1 Proposed Methods

### 3.1.1 Decision Tree

We train the DT using scikit-learn, with a maximum depth of fifteen and a maximum number of leaves of one thousand, using a train/test split of eighty percent and twenty percent, respectively. Data distribution with Label 0 is "Benign" and with Label 1 is "Attack" and Data distribution for DOS or DDOS with Label 0 is "Benign" and with Label 1 is "Attack". After training and testing with the DT, we export model parameters left children, right children, threshold, features, and value of each DT.

### 3.1.2 Decision Tree in eBPF

After getting the parameters of the DT model, we write an eBPF program to process the incoming malicious packets. In this eBPF program, we created eBPF maps to store the parameters of the DT model like left children, right children, threshold, features, value, etc. The eBPF program code is loaded and combined with other necessary code, such as maps and tables. eBPF maps store left and right children, thresholds, features, and value parameters and can access data within the eBPF program. The eBPF program, along with its associated maps, is loaded into the kernel. The program is attached to a specific hook or event point within the kernel socket. The socket where it sends and receives packets from network interfaces. Each captured packet is passed to the eBPF program.

### 3.1.3 Random Forest model

The RF algorithm is employed for training and testing on the overall dataset and DoS/DDoS detection performance. Model parameters such as left children, right children, threshold, features, and values of each DT are exported after training and testing with RF. The proposed RF algorithm is detailed in Algorithm 2, emphasizing its integration with eBPF for efficient intrusion detection.

### 3.1.4 Random Forest algorithm in eBPF

After getting the parameters of the RF model, we write an eBPF program to process the incoming malicious packets. In this eBPF program, we created eBPF maps to store the parameters of the RF model like Left Children, Right Children, Threshold, Features, Value of each DT, etc. The eBPF program code is loaded and combined with other necessary code, such as maps and tables. eBPF maps store left and right children, thresholds, features, and value parameters and can access data within the eBPF program. The eBPF program, along with its associated maps, is loaded into the kernel. The program is attached to a specific hook or event point within the kernel, such as a socket. The socket where it sends and receives packets from network interfaces. Each captured packet is passed to the eBPF program, which is attached to the socket for packet filtering .

### 3.1.5 SVM and TwinSVM

We trained machine learning algorithms, SVM and TwinSVM in Python on the CICIDS-2017 dataset with top 10 features. Data distribution with Label 0 is "Benign" and with Label 1 is "Attack" is shown in Figure 10 and Data distribution for DOS/ DDOS with Label 0 is "Benign" and with Label 1 is "Attack" . Then, we deploy the models in eBPF and userspace to compare the performance regarding the number of packets processed in a timeframe. One of the main hurdles in deploying these algorithms was representing the weights, which are floating point numbers in eBPF. eBPF does not allow floating-point numbers, so instead, we use a fixed-point representation of these floating-point numbers, which is nothing but multiplying all the weights with some large values (in our case, 216). This method has no problem for linear SVM and TwinSVM since they compare distances using these weights. However, Neural Networks (NN) and XGBoost inherently operate continuous values for outputs and later convert them into classes using some function like sigmoid or tanh. The final output is some function of the weights with

some non-linear transformation, so we cannot use fixed point representation in NN and XGBoost. The proposed SVM and TwinSVM algorithms are given in Algorithm 3 and Algorithm 4.

### 3.1.6 SVM and TwinSVM in eBPF

We implemented SVM and TwinSVM with a different kind of training, wherein we split the dataset into batches of 100 and trained a separate model for each of the 100 data points. Then we pick the best models of these and take a mean of their weights. We assign these weights to a new model, and it outperforms the top models by a slight margin over the validation set.

## 3.2 Architecture

### 3.2.1 eBPF

eBPF is a revolutionary technology with origins in the Linux kernel that can run sandboxed programs in a privileged context such as the operating system kernel. It is used to safely and efficiently extend the capabilities of the kernel without requiring to change kernel source code or load kernel modules.

Historically, the operating system has always been an ideal place to implement observability, security, and networking functionality due to the kernel's privileged ability to oversee and control the entire system. At the same time, an operating system kernel is hard to evolve due to its central role and high requirement towards stability and security. The rate of innovation at the operating system level has thus traditionally been lower compared to functionality implemented outside of the operating system.

eBPF changes this formula fundamentally. It allows sandboxed programs to run within the operating system, which means that application developers can run eBPF programs to add additional capabilities to the operating system at runtime. The operating system then guarantees safety and execution efficiency as if natively compiled with the aid of a Just-In-Time (JIT) compiler and verification engine. This has led to a wave of eBPF-based projects covering a wide array of use cases, including next-generation networking, observability, and security functionality.

Today, eBPF is used extensively to drive a wide variety of use cases: Providing high-performance networking and load-balancing in modern data centers

and cloud native environments, extracting fine-grained security observability data at low overhead, helping application developers trace applications, providing insights for performance troubleshooting, preventive application and container runtime security enforcement.and much more. The possibilities are endless, and the innovation that eBPF is unlocking has only just begun.

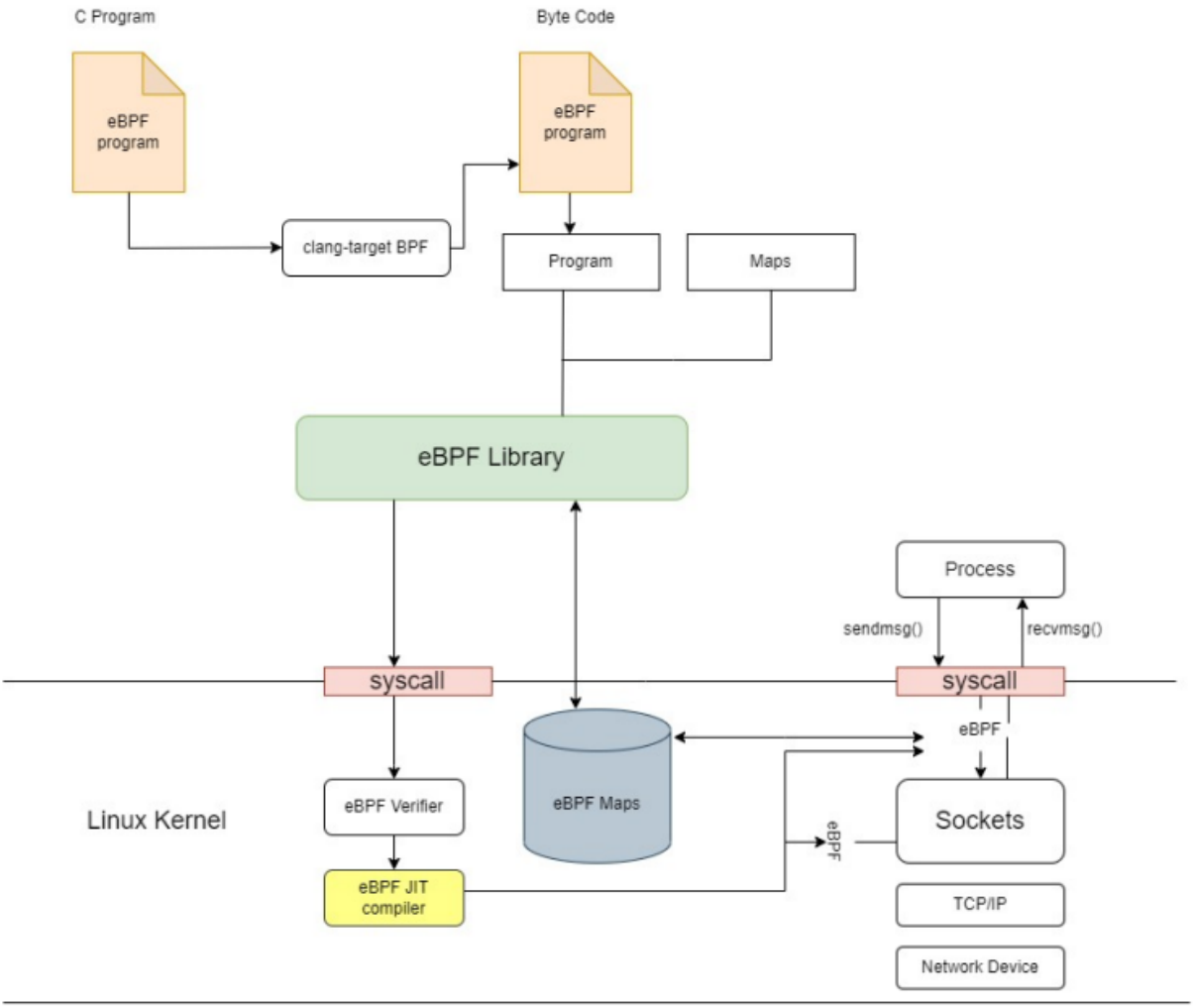


Figure 3.1: eBPF Architecture

### 3.2.2 XDP Design :

The design of the eXpress Data Path (XDP) is focused on enabling high-performance packet processing that can integrate cooperatively with the operating system kernel, while ensuring the safety and integrity of the rest of the system . XDP achieves this by defining a limited execution environment in the form of a virtual machine running eBPF code, which executes custom programs directly in kernel context before the kernel itself touches the packet data. This enables custom processing, including redirection, at the earliest possible point after a packet is received from the hardware. The kernel ensures the safety of the custom programs by statically verifying them at load time, and programs are dynamically compiled into native machine instructions to ensure high performance. XDP can be thought of as a "software offload" that offloads performance-sensitive processing to increase performance, while applications otherwise interact with the regular networking stack. XDP programs that don't need to access kernel helper functions can be offloaded entirely to supported networking hardware.

The diagram shows that XDP defines a limited execution environment in the form of a virtual machine running eBPF code, which executes custom programs directly in kernel context before the kernel itself touches the packet data. This enables custom processing, including redirection, at the earliest possible point after a packet is received from the hardware. The diagram shows that XDP can be thought of as a "software offload" that offloads performance-sensitive processing to increase performance, while applications otherwise interact with the regular networking stack. The diagram also shows that XDP programs that don't need to access kernel helper functions can be offloaded entirely to supported networking hardware. The diagram also shows that different eBPF programs can communicate with each other and with userspace through the use of BPF maps, which are a shared memory area that can be accessed by both kernel and userspace code. The diagram only shows the ingress path, but XDP can also be used on the egress path

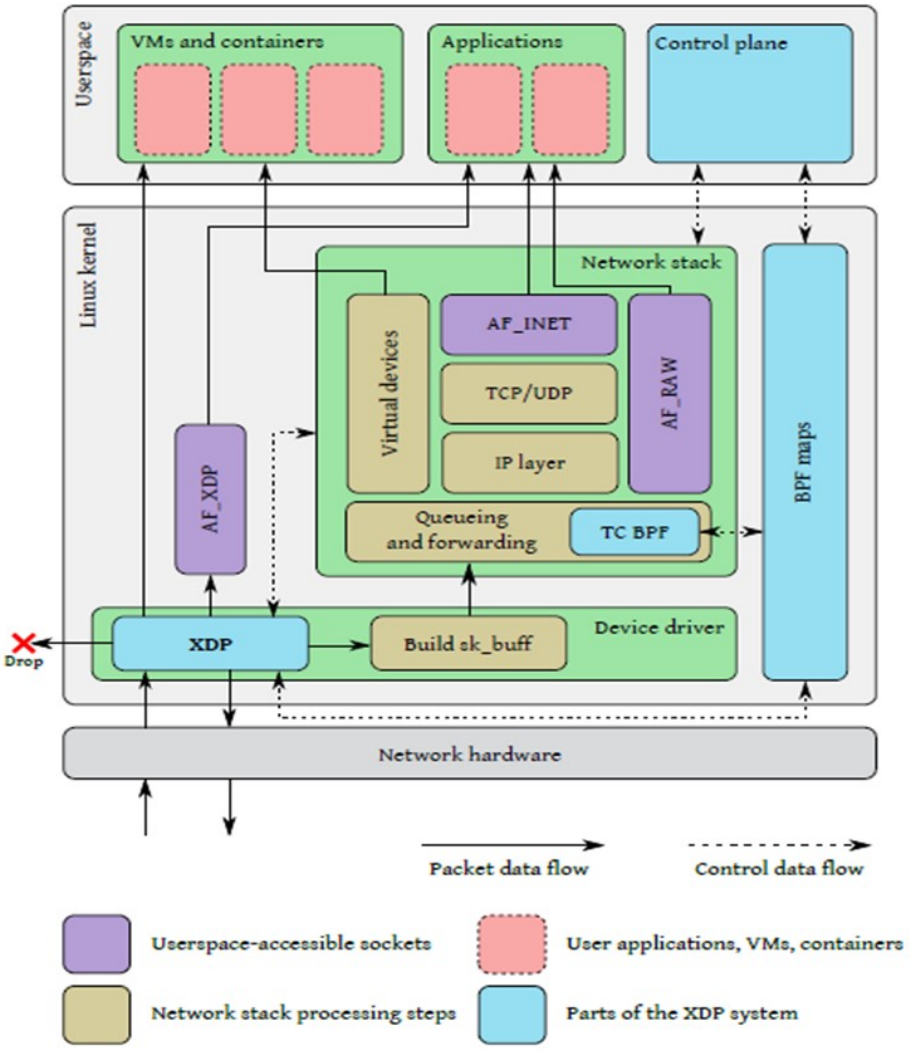


Figure 3.2: XDP Architecture

# Chapter 4

# Methodology

The steps involved in this process are:

## 4.1 Data Collection and Data Description

Utilizing information from the well-known CIC-IDS-2017 dataset and DoS/DDoS attack of the CIC-IDS-2017 dataset.

The CIC-IDS-2017 dataset is a collection of network traffic data consisting of 78 features. These features are destination port, total forward packets, total backward packets, minimum packet length, etc. This dataset is commonly used for testing intrusion detection systems.

## 4.2 Data Pre-Processing

The pre-processing steps involve data transmission (reading data from files), cleaning (infinity, NULL values, etc.), and reduction (size) when the raw data is collected and divided into train and test data.

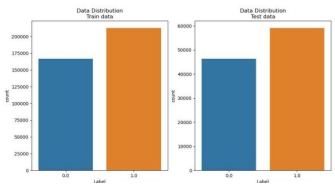


Figure 4.1: Data Distribution in dos ddos dataset

## 4.3 Feature Selection

The top 15 features that have a more significant influence on the target variable can be found by using the ANOVA F-test.We decreased to 10 features since SVM and TWSVM require more time, given that there are 78 features in all in the data set.

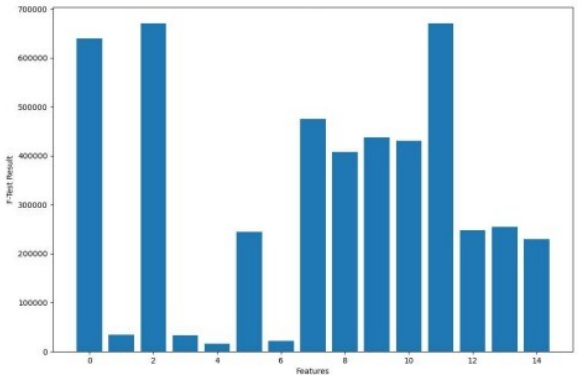


Figure 4.2: Features of Anova F-Test

## 4.4 Evaluation Metrics

Evaluated the performance using metrics including accuracy, precision, and recall.

# Chapter 5

# Implemention

## 5.1 Importing Libraries and Data Loading

For data preprocessing, feature selection, model evaluation, and visualization, the code imports a number of libraries. It contains machine learning task-specific decision tree and random forest classifiers, along with tools for scaling, encoding, and normalizing data. A concentration is placed on measures such as accuracy, precision, recall, and confusion matrices, with several neural network-related libraries written out. In addition, the code use seaborn to create confusion matrix figures and hides warnings.

## 5.2 Features Extracting Using Annova F-Test

To choose the top k features from the training dataset, the selectKBest function applies the ANOVA F-test. It uses the f_classif scoring function in conjunction with SelectKBest from the scikit-learn library. The function applies the F-test, extracts the selected features, and the target variable from the training and test sets. By concentrating on the most important variables, the resulting list, new_features, comprises the K best features as identified by the F-test, offering a simplified method to improve model training. By keeping only the most instructive properties, this function helps with feature selection for machine learning tasks, enhancing model performance.

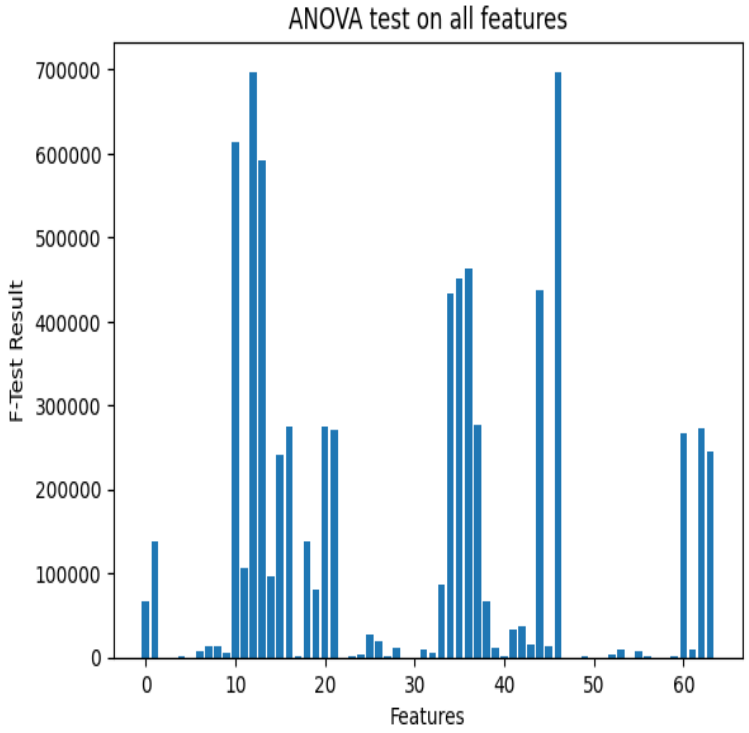


Figure 5.1: Annova F-test

## 5.3 Test and Train Data Visualization

The CIC 2017 data set is divided into a 1:4 ratio.

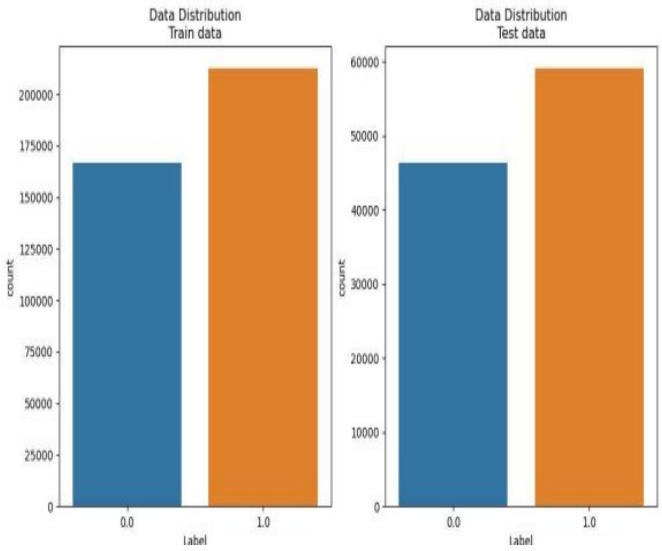


Figure 5.2: Annova F-test

## 5.4 Random Forest Model and Associated Confusion Matrix

X_train and Y_train are used to train the classifier. On the test set (x_test), predictions are made, and a number of performance measures are computed, such as specificity, accuracy, recall, precision, and F1-score. These measures provide a thorough evaluation of the classifier's predictive performance, emphasizing elements like total accuracy, accuracy in positive predictions, sensitivity to positive examples, and the harmonic mean of precision and recall. To calculate specificity, an analysis of the confusion matrix is conducted. The outcomes are then printed to provide a thorough analysis of the Random Forest model's performance.

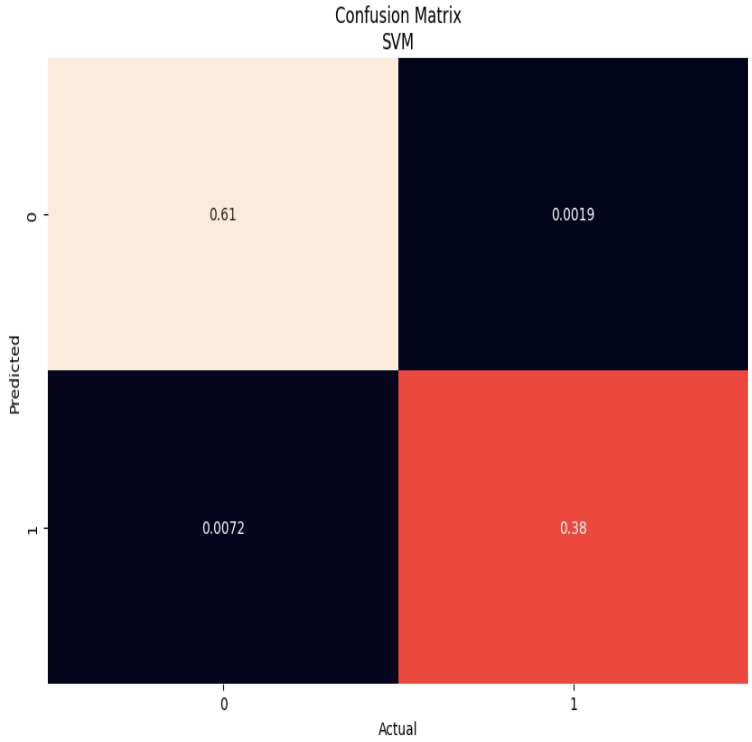


Figure 5.3: Random Forest Model

## 5.5 Decision Tree Model and Associated Confusion Matrix

Decision Tree classifier used to differentiate between "Normal" and "DoS" attacks using a dataset. The training set of data is used to fit the model, and predictions are produced for the test set (x_test). Afterwards, a number of performance metrics are calculated to evaluate the efficacy of the classifier, including accuracy, precision, recall, F1-score, and specificity. To determine true negatives (tn), false positives (fp), false negatives (fn), and true positives (tp), the confusion matrix is employed. The ratio of real negatives to all actual negatives is used to compute specificity. The outcomes are then printed to give a thorough assessment of how well the Decision Tree model detects "Normal" and "DoS" attacks.

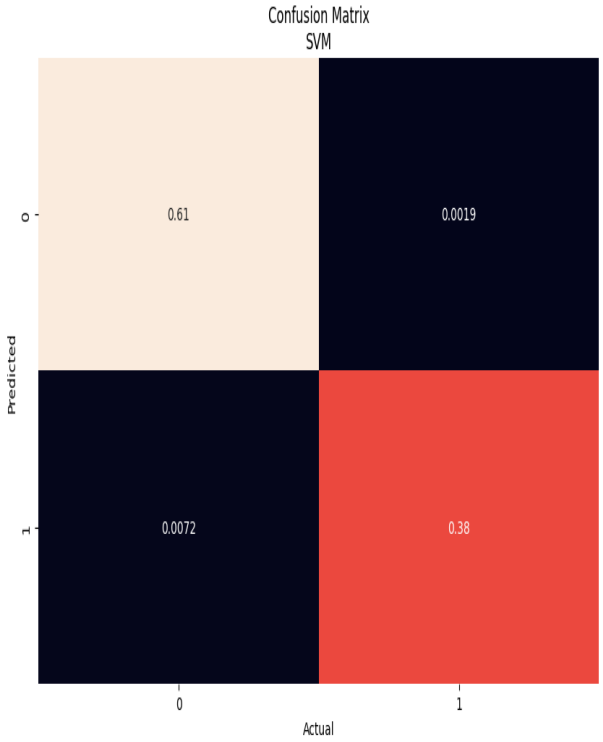


Figure 5.4: Decision Tree Model

## 5.6 SVM and Associated Confusion Matrix

Using batches of data, the code trains a linear kernel in a linear support vector machine (SVM). Each segment of the training data is iterated over by the loop, which fits a linear SVM with a regularization parameter of C=0.00001 after each iteration. The model is trained on the current batch for each iteration, and predictions are produced on the training and test sets.For both training and test sets, performance metrics like as accuracy, precision, and recall are computed. The coefs_intercepts list then contains these metrics as well as the SVM model's coefficients and intercept. Accuracy, precision, and recall for both the training and test sets are displayed in real-time along with the training progress.

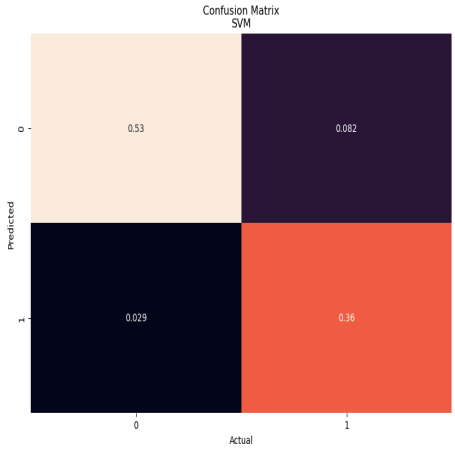


Figure 5.5: SVM Model

## 5.7 TWSVM and Associated Confusion Matrix

Using the test set (x_test), Twin Support Vector Machine (TwinSVM) model (twsvm) is used in this code to make predictions. It then computes the accuracy, precision, recall, and F1-score to assess the model's performance. To calculate specificity, the confusion matrix is used to extract true negatives (tn), false positives (fp), false negatives (fn), and true positives (tp). The final findings, which offer a thorough evaluation of the TwinSVM model's performance in binary classification, are printed. The code has redundant print statements for correctness, precision, and recall that might be eliminated to make it more concise.

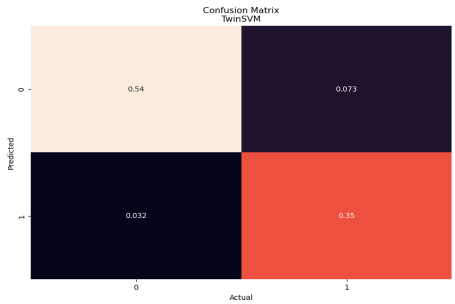


Figure 5.6: TWSVM Model

## 5.8 XDP program to filter packets

```c
#include <linux/if_ether.h>
#include <linux/ip.h>

SEC("filter")
int xdp_filter(struct __sk_buff *skb) {
    void *data_end = (void *)(long)skb->data_end;
    void *data = (void *)(long)skb->data;

    // Access the Ethernet header
    struct ethhdr *eth = data;

    // Check if the packet is IP
    if (eth + 1 <= data_end && ntohs(eth->h_proto) == ETH_P_IP) {
        // Access the IP header
        struct iphdr *ip = data + sizeof(struct ethhdr);

        // Check if the packet has a specific source IP address
        if (ip + 1 <= data_end && ip->saddr == htonl(0x12345678)) {
            // Drop the packet
            return XDP_DROP;
        }
    }

    // Allow the packet to pass through
    return XDP_PASS;
}
```

Figure 5.7: Using XDP Packet Filtering

The xdp_filter function is the main entry point for the XDP program. The struct __sk_buf f structure provides access to the packet data and metadata. We check if the packet is an IP packet and if it has a specific source IP address. and if the condition is met, the function returns XDP_DROP to drop the packet; otherwise, it returns XDP_PASS to allow the packet to pass through.

# Chapter 6

# Results and Discussions

## 6.1 Experimental Results

**Random Forest:** Compared to DT and SVM models, RF is performing better. This can be seen with the accuracy of 99.44 percent. Also, we can see in the table that we calculated DoS/DDoS detection, and the accuracy was found to be 99.57 percent.

| Performance Parameters | Method | DT | RF | SVM | TwinSVM |
|---|---|---|---|---|---|
| Accuracy | train | 99.52 | 99.59 | 88.77 | 93.87 |
| | test | 99.38 | 99.44 | 88.74 | 93.82 |
| Precision | train | 99.71 | 99.74 | 78.97 | 98.58 |
| | test | 99.46 | 99.51 | 78.97 | 98.49 |
| Recall (or) Sensitivity | train | 98.49 | 98.72 | 73.41 | 75.9 |
| | test | 98.21 | 99.37 | 73.26 | 75.78 |
| F1 Score | train | 99.09 | 99.23 | 76.09 | 85.76 |
| | test | 98.83 | 98.94 | 76.01 | 85.65 |
| Specificity | train | 99.89 | 99.91 | 93.71 | 99.65 |
| | test | 99.81 | 99.82 | 93.73 | 99.63 |

Table 6.1: Experimental results of overall (all packets inspection) CIC-IDS-2017 dataset in userspace

- A virtual machine called eBPF uses bytecode to execute applications without changing the source code of the kernel. The userspace implementation is slower than that of the eBPF. Utilizing the CICIDS-2017 dataset, sci-kit-learn was used to train using DT, RF, SVM, and

| Performance Parameters | Method | DT | RF | SVM | TwinSVM |
|---|---|---|---|---|---|
| Accuracy | train | 99.73 | 99.82 | 84.43 | 88.49 |
| | test | 99.57 | 99.58 | 84.43 | 88.42 |
| Precision | train | 99.87 | 99.88 | 84.28 | 98.53 |
| | test | 99.71 | 99.65 | 84.31 | 98.49 |
| Recall (or) Sensitivity | train | 99.56 | 99.80 | 86.74 | 79.42 |
| | test | 99.52 | 99.60 | 86.70 | 79.33 |
| F1 Score | train | 99.76 | 99.84 | 85.49 | 87.95 |
| | test | 99.62 | 99.62 | 85.48 | 87.88 |
| Specificity | train | 99.83 | 99.85 | 81.83 | 98.67 |
| | test | 99.64 | 99.55 | 81.88 | 98.63 |

Table 6.2: Experimental results of DoS/DDoS of CIC-IDS-2017 dataset in userspace

TwinSVM with a test/train split of 1:4. Our experimental findings demonstrate that our suggested machine learning algorithms, DT, RF, SVM, and TwinSVM, perform more accurately than previous comparable work: 88.73, 93.82, 99.38, and 99.44.

| Algorithms | Userspace (Mean) | eBPF (Mean) |
|---|---|---|
| Decision Tree packet/s | 46239 | 109691 |
| Random Forest packet/s | 45978 | 108534 |
| SVM packet/s | 45590 | 92978 |
| TwinSVM packet/s | 38430 | 109865 |

Table 6.3: Time taken in user space vs kernel space of overall (all packets inspection) CIC-IDS-2017 dataset

| **Algorithms** | **Userspace (Mean)** | **eBPF (Mean)** |
|---|---|---|
| Decision Tree packet/s | 42463 | 106421 |
| Random Forest packet/s | 41632 | 105245 |
| SVM packet/s | 49376 | 92581 |
| TwinSVM packet/s | 42487 | 117536 |

Table 6.4: Time taken in user space vs kernel space of DoS/DDoS of CIC-IDS-2017 dataset

| **Author/Parameter** | **Algorithm** | **Accuracy** | **User Space packet/s** | **eBPF packet/s** |
|---|---|---|---|---|
| Proposed | Decision Tree | 99 | 125420 | 152274 |
| Proposed | Decision Tree | 99.38 | 46239 | 109691 |
| Proposed | Random Forest | 99.44 | 45978 | 108534 |
| Proposed | SVM | 88.74 | 45590 | 92978 |
| Proposed | TwinSVM | 93.82 | 38430 | 109865 |

Table 6.5: Accuracy, user space, eBPF space comparison with related work

# Chapter 7

# Conclusions and future works

## 7.1 Conclusion

The performance of the Intrusion Detection System has been increased using XDP and eBPF together, the results shows that the packet processing per seconds has been increased to about 50% in kernel space than in the user space.

## 7.2 Future Work

The future scope of this work extends to exploring the integration of various machine learning algorithms with XDP and eBPF to elevate the performance and accuracy of Intrusion Detection System.

# Bibliography


[1] N. ANAND, M. SAIFULLA, and P. K. Aakula, "High-performance intrusion detection systemusing ebpf with machine learning algorithms," 2023.

[2] M. A. Vieira, M. S. Castanho, R. D. Pacífico, E. R. Santos, E. P. C. Júnior, and L. F. Vieira, "Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–36, 2020.

[3] S.-Y. Wang and J.-C. Chang, "Design and implementation of an intrusion detection system by using extended bpf in the linux kernel," *Journal of Network and Computer Applications*, vol. 198, p. 103283, 2022.

[4] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, "The express data path: Fast programmable packet processing in the operating system kernel," in *Proceedings of the 14th international conference on emerging networking experiments and technologies*, pp. 54–66, 2018.

[5] M. Aljanabi, M. A. Ismail, and A. H. Ali, "Intrusion detection systems, issues, challenges, and needs," *International Journal of Computational Intelligence Systems*, vol. 14, no. 1, pp. 560–571, 2021.

[6] M. Bachl, J. Fabini, and T. Zseby, "A flow-based ids using machine learning in ebpf," *arXiv preprint arXiv:2102.09980*, 2021.

[7] H. Wieren, "Signature-based ddos attack mitigation: Automated generating rules for extended berkeley packet filter and express data path," 2019.

[8] R. Panigrahi and S. Borah, "A detailed analysis of cicids2017 dataset for designing intrusion detection systems," *International Journal of Engineering & Technology*, vol. 7, no. 3.24, pp. 479–482, 2018.

[9] E. Leblond and P. Manev, "Introduction to ebpf and xdp support in suricata," *WP-eBF-XDP-092021–1, Stamus Networks*, 2021.

[10] B. Packet, "Xdp-programmable data path in the linux kernel,"