Architecture
Image Studies

Vol.6, Issue 4, pp.855-878, 2025
DOI: https://doi.org/10.62754/ais.v6i4.692

https://journals.ap2.pt/index.php/ais/index

Evaluation of SDN Flow Table Manipulation Attack Using Machine Learning Techniques

Kamal Singh[1], Brijesh Kumar[2]

Abstract

Security vulnerabilities in Software-Defined Networking (SDN) can expose the entire network to serious cyberattacks. This study focuses on Flow Rule Manipulation, a critical threat in which an attacker adds, modifies, or deletes flow rules in the SDN controller. Manipulating flow rules can change the forwarding logic, disrupt network traffic, steal data, or redirect sensitive information to malicious hosts. In this study, we simulate a real-time flow rule manipulation attack on an SDN controller to examine its effects on the network. The attack is executed through a structured sequence that includes reconnaissance and active manipulation in a live SDN environment with an active data plane. By inserting malicious flow rules, the attacker can divert traffic, intercept data, and alter network behaviour. The study employs various machine learning techniques to detect attempts at manipulating flow tables and assesses which models are most effective in identifying abnormal changes. Several machine learning algorithms are evaluated, including Logistic Regression, Random Forest, Support Vector Machine (SVM), Gradient Boosting, and K-Nearest Neighbors (KNN), to determine the most effective detection method. The results demonstrate the potential of machine learning-based detection to enhance the security and resilience of modern critical network against flow rule manipulation attacks.

Keywords: *SDN, SDN Security, OpenFlow, Network Security, Machine Learning, SDN Flow Attacks.*

Introduction

Software-Defined Networking has emerged as a modern era networking technology due to its centralised control architecture and programmability. It is widely deployed in cloud platforms, telecommunications networks, and enterprise environments. SDN enables virtualisation, automation, and scalable services, making it suitable for modern applications such as 5G and the Internet of Things (IoT). As SDN adoption increases, securing the architecture becomes essential.

In the centre of SDN is the controller, which manages the entire network and determines how traffic should flow. Since it exercises full control over the forwarding plane, the controller represents a critical point of vulnerability. Attackers can target its Application Programming Interfaces (APIs), applications, and data plane connexions to modify the behaviour of the network or disrupt communication. Any compromise of the controller can affect the entire SDN environment.

In this study, we used the Ryu controller [2] with the Mininet emulator [3] and OpenFlow switches [4] to investigate one of the most significant threats: flow rule manipulation. This attack alters the forwarding logic by adding, modifying, or deleting flow rules in the controller. According to the STRIDE threat model, the manipulation of flow rules falls under the tampering category, along with code injection and priority-bypass attacks. A malicious rule can redirect traffic to a rogue server, intercept sensitive data, or disrupt normal communication.

Unlike volumetric Distributed Denial-of-Service (DDoS) attacks, flow manipulation does not generate large amounts of traffic. It operates silently and can appear as a normal network issue. Consequently, traditional monitoring tools that rely on traffic volume may not detect it. The primary indication may be changes in the flow table, making early detection difficult without adequate inspection mechanisms.

To address this challenge, our work evaluates flow rule manipulation attacks using several machine learning techniques. Machine learning plays a key role in identifying abnormal controller behaviour and can detect subtle changes that indicate tampering. Traditional SDN security relies on

[1] FET, MRIIRS, kamallohiaji@gmail.com (corresponding author).
[2] MRIIRS, Email - brijesh.set@mriu.edu.in.

static, rule-based systems, such as threshold alerts or predefined conditions. However, these systems fail when attackers replicate patterns of normal traffic or when attacker behaviour evolves over time. Machine learning addresses these limitations by learning from real network data. It can differentiate between normal and malicious activities, even if attackers change their methods. The study applies several machine learning models Logistic Regression, Random Forest, Support Vector Machine (SVM), Gradient Boosting, and K-Nearest Neighbours (KNN) to evaluate their effectiveness in detecting flow rule manipulation attacks.

The remainder of this paper is organised as follows. Section II provides an overview of SDN. Section III reviews related work. Section IV explains the methodology and experimental setup. Section V presents the results, and Section VI concludes with key findings and future research directions.

Background

SDN Architecture

The Open Networking Foundation (ONF) outlines a hierarchical architecture for SDN that comprises three main layers:

Infrastructure Layer: This layer comprises physical and virtual network elements, such as switches, routers, gateways, and servers. These network elements are responsible for forwarding data packets.

Control Layer: This layer contains the SDN controller, which manages and controls the network elements in the Infrastructure Layer. The SDN controller communicates with, manages, and controls network devices using a southbound Application Programming Interface (API), such as OpenFlow, and a northbound API, such as Representational State Transfer (REST), for application interaction.

Application Layer: This layer comprises the applications that utilise the network services provided by the SDN solution.

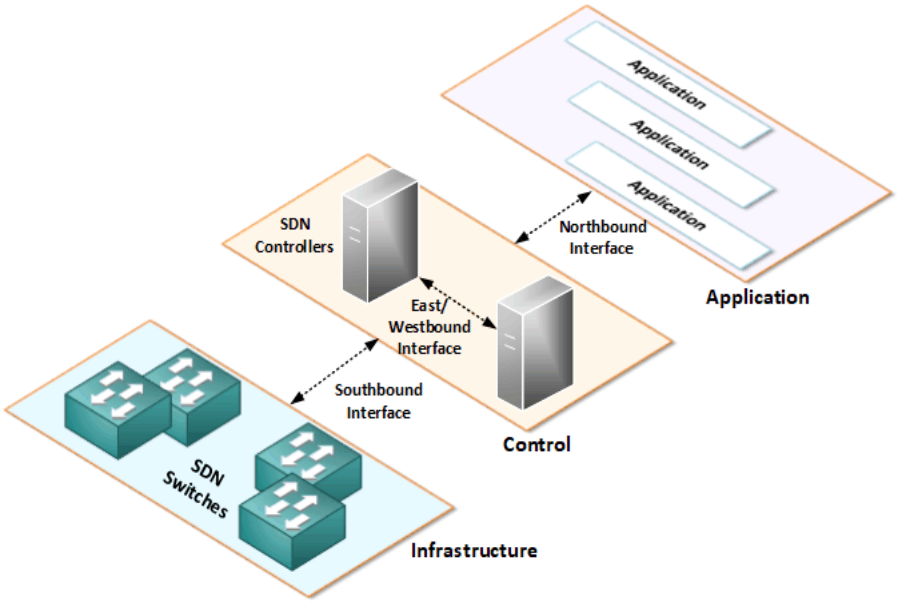

Figure 1: ONF SDN Architecture

B. SDN Controller

The OpenFlow SDN controller manages and controls data traffic flow between network devices, serving as an essential element of an SDN network. Open Network Operating System (ONOS) and OpenDaylight (ODL) are widely used open-source network operating systems or controllers. The SDN controller encompasses essential components that contribute to its functionality:

Southbound API: It provides communication between the OpenFlow controller and network devices, enabling efficient message exchange, traffic handling, and forwarding.

Northbound API: It connects the OpenFlow controller to applications, enabling them to request network services such as load balancing, security, and Quality of Service.

East/Westbound API: It facilitates communication between SDN controllers and distributed controllers, promoting coordination and information exchange among network-distributed controllers.

Security Module: The OpenFlow SDN controller ensures network security by providing user authentication, encryption, and access control to authorised users only.

Management Module: Configures and manages the OpenFlow controller, network devices, and applications, ensuring operational efficiency in the SDN ecosystem.

Literature Review

Research on SDN security consistently shows that the programmability that makes SDN attractive also expands its attack surface across controllers, APIs, and inter-plane links. Systematic mappings of these threats highlight recurring issues across controller designs, classifying risks according to confidentiality, integrity, and availability. The controller remains a single point of failure, and DDoS represents a major concern [25]. Beyond broad taxonomies, work on the topology discovery service explains how link fabrication and host-location hijacking mislead the controller's view, enabling man-in-the-middle and denial-of-service attacks, and outlines countermeasures to secure host tracking and link discovery [22].

In the area of detection, several studies integrate real-time telemetry with machine learning. Security Information and Event Management (SIEM) pipelines combined with machine learning improve accuracy and reduce false positives by correlating events and adapting to live traffic from diverse attacks [18]. Two-stage detectors that pair fast statistical checks with machine learning classifiers reduce noise while maintaining sensitivity. Entropy-based triggers followed by machine learning verification represent a common design for early DDoS alerts [16]. Deep-learning approaches tailored to flow-time-series data such as optimised one-dimensional Convolutional Neural Network (1D-CNN) models strengthened by the Non-dominated Sorting Genetic Algorithm II (NSGA-II) and the protocol-level Synthetic Minority Over-sampling Technique (SMOTE) achieve near-perfect accuracy on emulated SDN datasets [8].

Architectural placement also influences effectiveness. Hybrid solutions divide tasks between the data plane and the controller or edge. Programming Protocol-independent Packet Processors (P4) switches handle lightweight screening, while more advanced edge models complete classification, providing fast reaction times without sacrificing precision [15]. In IoT-based SDN environments, deploying XGBoost-style models at the edge reduces controller load and improves response time. With careful tuning and cross-validation, these setups maintain over 99% accuracy on Canadian Institute for Cybersecurity Intrusion Detection System (CICIDS)-type datasets, with alerts enforced through the SDN controller [10]. Traditional supervised methods such as SVM, KNN, Decision Trees (DT), Random Forest (RF), and Naive Bayes (NB) remain widely used. Studies using CICIDS2017/2019 and emulated environments demonstrate that SVM often excels in recall, while KNN and RF perform strongly, depending on the feature sets and traffic composition, especially when integrated with testbeds and Network Intrusion Detection Systems (NIDS) for live mitigation [24].

Domain-specific deployments motivate more specialised defences. In SDN-Unmanned Aerial Vehicle (SDN-UAV) systems, ensemble learning combined with honeypot UAVs enhances robustness and provides continuous, labelled data for retraining. Federated controller layouts further mitigate single points of failure [17]. Other UAV-focused studies combine Genetic Algorithm (GA)-based placement, shortest-path controller selection, and Feedforward Neural Network (FNN)-based DDoS detection with Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) feature reduction to enhance throughput while maintaining modest detection cost [23].

Proactive and explainable pipelines are increasingly emphasised. Predictive Intrusion Detection System (IDS) frameworks built using Random Forest enable SDN controllers to act proactively before damage occurs in 5G-like environments, although accuracy still depends on the dataset coverage [11]. Broader surveys indicate a shift toward deep learning and hybrid deep learning-machine learning stacks, which aim to balance feature learning with fast inference, while emphasising the need for solutions that handle time-varying traffic, scalability, and context integration [12]. Comparative analyses of SDN controllers reveal differences under DoS conditions: ONOS generally offers lower latency but may experience higher loss. At the same time, ODL maintains availability at the cost of slower response times, raising questions about controller choice and motivating the adoption of machine-learning-

assisted mitigation strategies [19], [20]. At the embedded edge, Decision Trees and SVM models achieve high accuracy, with ODL handling heavier workloads, Ryu offering simplicity, and ONOS providing advanced security mechanisms [13]. Long Short-Term Memory (LSTM) and linear SVM pipelines tied to SDN controllers demonstrate real-time anomaly detection and automated blocking. They can be extended to Virtual Private Network (VPN)-enabled overlays for secure communication [9].

Finally, threat modelling and governance continue to play a critical role. STRIDE-based analyses outline spoofing, tampering, information disclosure, DoS, and privilege-escalation paths, along with practical safeguards such as authentication, authorisation, encryption, logging, and capacity-aware DoS monitoring [5]. Complementary reviews suggest combining traditional controls with artificial intelligence and moving-target defences, such as Internet Protocol (IP), Media Access Control (MAC), and port randomisation, to reduce attacker dwell time while keeping overhead manageable [7]. In beyond-5G scenarios, SDN-centric routing and distributed control are also studied for interference mitigation, with implications for secure and low-latency operation in dense wireless environments [14].

Methodology And Experimental Setup

Flow manipulation attacks in SDN can be identified through several abnormal flow-level behaviours. One indication is an unusually high number of flow-modification events, where the flow-mod count rises beyond normal operational levels. Another strong indication of tampering is the presence of flow entries with abnormally high priority values, which allows malicious rules to override legitimate forwarding policies. Rules with empty action fields are also suspicious, as they cause packets to be silently dropped rather than forwarded. Very short-lived flows that appear and disappear rapidly may indicate continuous rule insertion or deletion by an attacker. Additionally, a sudden increase in flows targeting specific destination ports suggests selective manipulation of network traffic.

The SDN testbed used for this study was constructed using the Ryu controller and Open vSwitch data plane devices. Normal traffic was generated using tools such as iPerf and Ping. During the attack phase, a malicious SDN application injected unauthorised flow-mod messages into the switch tables to simulate a flow-table manipulation attack. Throughout the experiment, the controller and switches logged every FlowMod, PacketIn, and FlowStats event to support subsequent analysis.

The flow rule manipulation attack was successfully executed, resulting in a clear and measurable impact on network availability. The attack was performed using the Ryu controller with OpenFlow switches, and real-time traffic was generated within the test environment. Kali Linux was used to launch the attack, with tools such as Nmap and cURL assisting in reconnaissance and rule injection.

Successful Reconnaissance: The initial Nmap scan and subsequent cURL-based enumeration confirmed that the Ryu controller's REST API was accessible without authentication, exposing a significant security misconfiguration. This allowed the attacker to gather information about the controller, switches, and network topology.

Rule Injection Success: The malicious flow entry was successfully accepted by the Ryu controller and installed into the flow table of the target switch (DPID 1). As designed, the controller propagated the rule to the data plane without validating the intent or security implications, demonstrating how unauthorised instructions can directly alter forwarding behaviour.

Effective Service Disruption: The injected rule caused a complete disruption of Secure Shell (SSH) connectivity. The SSH session repeatedly timed out because the Transmission Control Protocol (TCP) handshake packets were silently dropped. This demonstrated how a single malicious API call can create a targeted denial-of-service condition against a critical administrative service.

Stealth of the Attack: Unlike volumetric DDoS attacks, the flow manipulation attack does not generate high traffic volumes. It remains silent and difficult to detect using traffic-volume-based monitoring tools. The resulting network failure appears to be a normal operational fault, making accurate diagnosis challenging without examining the switch's flow entries.

Machine Learning Evaluation: Machine learning techniques were applied to analyse and classify the captured flow data, enabling the detection of abnormal flow behaviour created during the attack. Multiple machine learning algorithms, including Logistic Regression, Random Forest, SVM, Gradient Boosting, and KNN, were employed to identify the most effective detection approach.

Experimental Setup

The experimental environment was constructed using the Ryu controller in combination with Open vSwitch devices emulated through Mininet. The controller operated on an Ubuntu 20.04.1 LTS system, while the attack machine was implemented using a Kali Linux 2024.2 virtual machine. Mininet version 2.3.1b4 was used to emulate network topology and OpenFlow-compatible switches.

Normal network traffic was generated using tools such as iPerf and Ping to create realistic communication patterns within the testbed. During the attack phase, a malicious SDN application running on the attacker's machine injected unauthorised FlowMod messages into the switch flow tables to simulate flow rule manipulation attack.

Attack Simulation

This section describes the attack simulation conducted in the SDN test environment.

1) Network Reconnaissance and Enumeration: In the first phase of the attack simulation, Nmap was used to identify open ports and assess the attack surface by scanning the network for reachable hosts and exposed services.

The nmap tool was used to perform a TCP SYN scan (-sS) against the target IP range. This scan successfully identified the host 192.168.31.3 as active and revealed that its port 8080, the default port for the Ryu controller's REST API, was open and accessible.

nmap -sS 192.168.31.0/24

Figure: Nmap Service Enumeration

Once the Ryu controller's API endpoint was confirmed to be accessible, the next phase involved enumerating the controller and its connected switches using cURL. The Ryu REST API provides multiple endpoints for retrieving network topology information and switch-level details. By querying these endpoints, the attack process gathered essential information about the programmable components of the SDN environment, including the list of active switches and their flow table descriptions.

The following curl command was used to request a list of all switches connected to the controller. This is typically achieved by querying the /stats/switches endpoint.

curl http://192.168.31.3:8080/stats/switches

Following this, a more detailed query was issued to gather specific information about a target switch with Datapath ID (DPID) 1, including its flow table entries and capabilities, via the /stats/desc/1 endpoint.

curl http://192.168.31.3:8080/stats/desc/1

2) Active Flow Rule Manipulation: The central phase of the attack involved injecting a malicious flow rule into the SDN switch to disrupt normal network operations deliberately. Using a cURL POST request to the /stats/flowentry/add endpoint, a new flow entry was installed on switch DPID 1 through the Ryu controller. The injected rule was configured with a very high priority to override legitimate forwarding entries, and its match fields targeted IPv4 packets using TCP with destination port 22, corresponding to SSH traffic. The action field was intentionally left empty, which in OpenFlow semantics

results in all matching packets being dropped. This malicious rule, therefore, forced the switch to silently discard SSH connection attempts, producing a targeted denial-of-service condition against SSH services in the network.

The malicious rule was crafted with the following parameters:

dpid: 1 – The target switch.

priority: 60000 – A very high priority to ensure this rule takes precedence over any existing permitting rules.

match – The rule was designed to match specific packet criteria:

eth_type: 0x0800 – IPv4 packets.

ip_proto: 6 – TCP protocol.

tcp_dst: 22 – Destination port 22 (SSH).

actions: [] – An empty action list. In the OpenFlow protocol, this signifies that matching packets should be dropped.

This rule effectively instructs the switch to silently discard all SSH connection attempts that pass through it, thereby creating a targeted denial-of-service attack against SSH traffic.

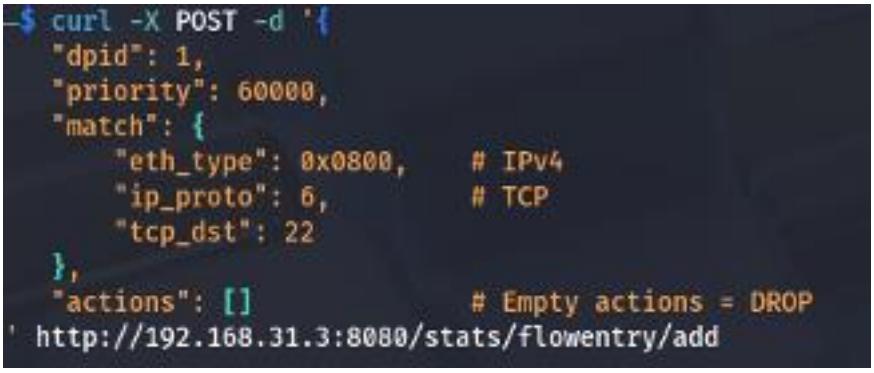

Figure: Flow Rule Addition

3) Verification of Attack Impact: To verify the success of the attack, an SSH connection attempt was initiated from the Kali Linux host to a target host that was reachable via the compromised switch. The connection request failed, as the malicious flow rule on the switch dropped the TCP SYN packets destined for port 22. This confirmed that the flow table manipulation was successful and that the network's behaviour had been maliciously altered, demonstrating a clear breach of security and availability.

Figure: Verification of Attack Impact

Verifying the Updated Flow Table Rule: The impact of the attack was further validated by examining the flow table entries before and after the malicious rule injection. Prior to the attack, the flow table displayed the standard default rules that the controller installs during normal operation. Once the attack was executed, the controller's FlowManager interface displayed only the malicious rule, indicating that the switch had successfully applied it.

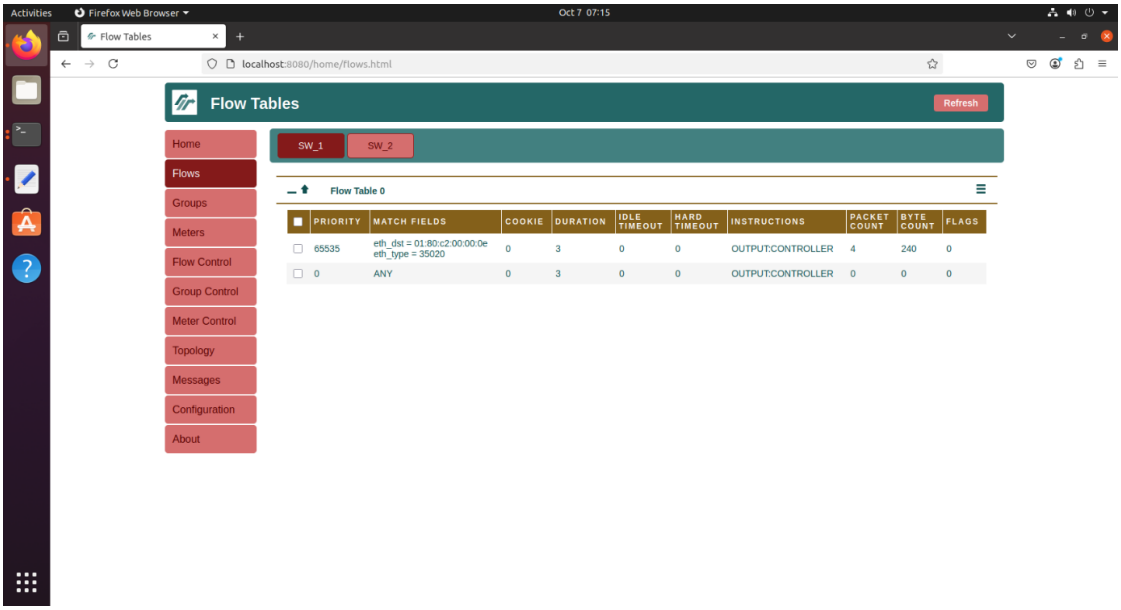

Figure: SDN Flow Table Before the Attack

The following screenshot shows the flow table after the attack simulation:

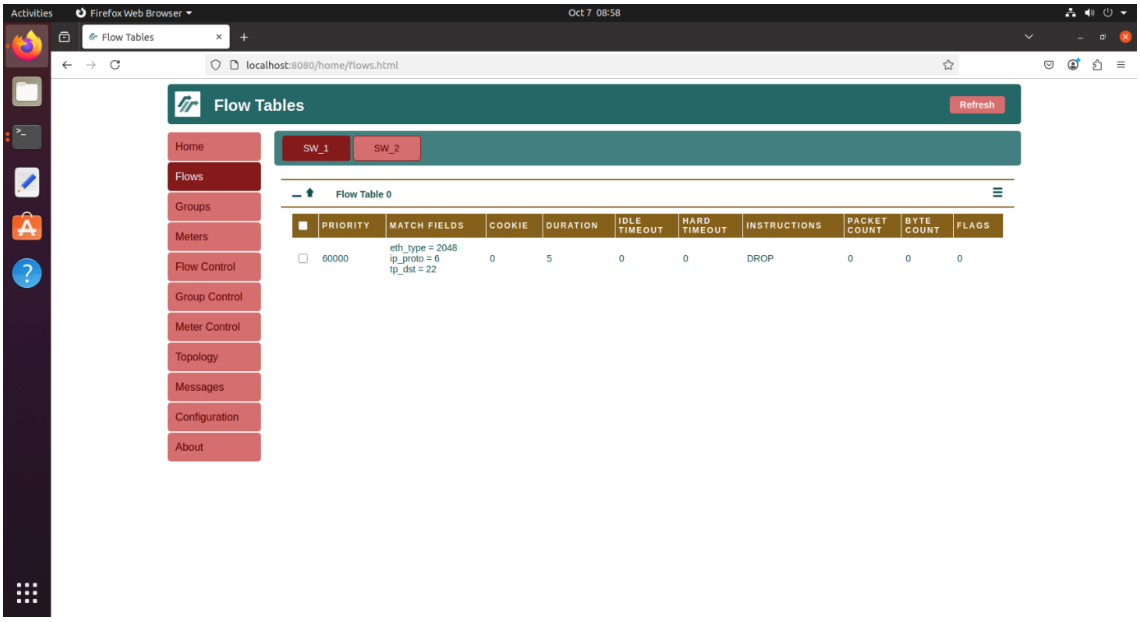

Figure: SDN Flow Table After the Attack

In the attack phase, the malicious flow rule was injected into the switch via a curl POST request, and it was assigned a very high priority to override legitimate forwarding entries and block the targeted SSH traffic. The flow manager's logs confirmed this.

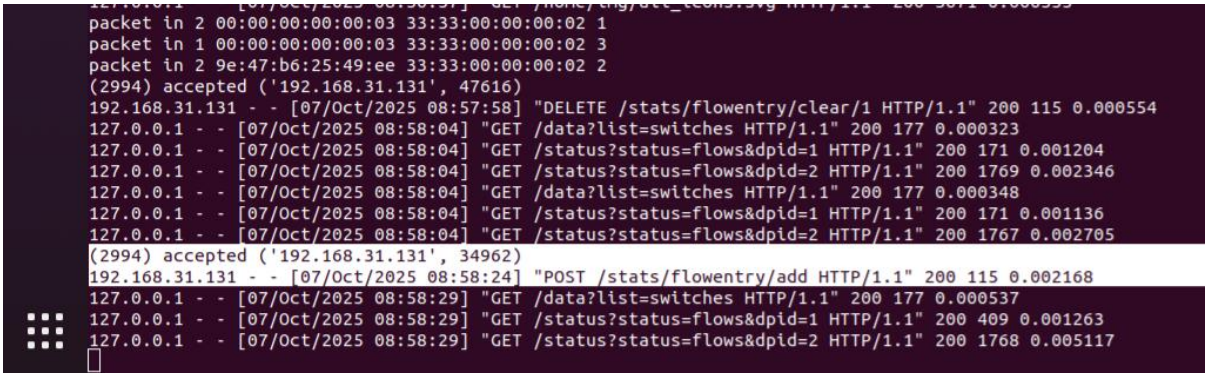

Figure: Flow Logs

Evaluation Of Sdn Attacks Using Ml Techniques

Selected Machine Learning Techniques

1) Logistic Regression: Logistic Regression is a machine learning technique used to predict whether an instance belongs to one category or another, such as normal or attack. It operates by taking input data, such as various flow variables, and combining them with learned weights. The algorithm then applies a sigmoid function to transform the result into a probability between 0 and 1. If the probability exceeds 0.5, the model predicts one class (attack), and if it is below 0.5, it predicts the other class (normal).

2) Random Forest: Random Forest is a supervised machine learning technique that learns from labelled data. It is used for both classification and regression tasks, though it is predominantly used for classification. This technique derives its name from its operational structure. It constructs a forest composed of many decision trees. Each decision tree learns in a slightly different manner by examining random subsets of the data and random features. When new data is provided to the model, each decision tree generates its own prediction. Subsequently, the forest aggregates the results through majority voting for classification tasks or averaging for regression tasks. Due to this randomness, the technique achieves greater accuracy and stability. Random Forest makes decisions based on the collective output of many decision trees.

3) Support Vector Machine: SVM is a supervised learning algorithm used primarily for classification tasks. In this case, SVM classifies instances as either normal behaviour or an attack. It operates by identifying the optimal boundary, either a line in two dimensions or a hyperplane in higher dimensions, that separates data points of two different classes. The objective is to draw this boundary with the widest possible margin between the two groups. Data points closest to the boundary are called support vectors. These support vectors help define the boundary's position. When new data is provided to the model, SVM determines on which side of the boundary the data point falls.

4) Gradient Boosting: Gradient Boosting is a supervised learning algorithm used for both classification and regression. It trains models using multiple decision trees. In essence, it combines many weak models, usually decision trees, step by step, to construct one strong model. Initially, the model generates predictions, which may be correct or incorrect. It then analyses errors across different trees and builds a new tree designed to correct them. This process continues iteratively, with each successive tree improving upon the previous one, until predictions become accurate.

5) K-Nearest Neighbours: KNN is a supervised learning algorithm used for both classification and regression. When the KNN model is given a new data point, it examines the k nearest points (neighbours) from the training data based on distance metrics. It then determines which class is most common among those neighbours and assigns that class to the new point.

Model Training and Testing Process

This section evaluates the flow rule manipulation attack using several machine learning techniques. The objective of this evaluation is to determine how effectively different algorithms can identify abnormal flow-level patterns created during the attack. After collecting both normal and attack traffic from the SDN testbed, the dataset was preprocessed and used to train five supervised learning

models: Logistic Regression, Random Forest, Support Vector Machine, Gradient Boosting, and K-Nearest Neighbours.

Each model was trained using the same set of features extracted from FlowStats, FlowMod, and PacketIn events. These features captured changes in flow counts, priority values, action fields, and packet-handling behaviour, which are key indicators of manipulation. The trained models were then tested on unseen data to measure their ability to distinguish normal behaviour from tampered flow entries.

Feature	**Description**
flow_mod_count	Number of flow modification messages per interval
priority	Assigned priority to flow rules
actions empty	Indicator for rules without forwarding actions
tcp_dst	Destination port of flow (e.g., 22, 80, 443)
ip_proto	Transport protocol (TCP, UDP, ICMP)
Packet_count / byte _count	Flow-level counters
Flow_duration	The time a flow rule stays active
src_port_entropy	Entropy of source ports observed

To compare model performance, several evaluation metrics were used, including accuracy, precision, recall, F1-score, and the area under the Receiver Operating Characteristic (ROC) curve. These metrics demonstrate how well each model can detect the attack under different conditions. Confusion matrics were also generated to show the distribution of true positives, false positives, false negatives, and true negatives for each technique. ROC curves and learning curves were plotted to examine detection patterns and training behaviour across the models.

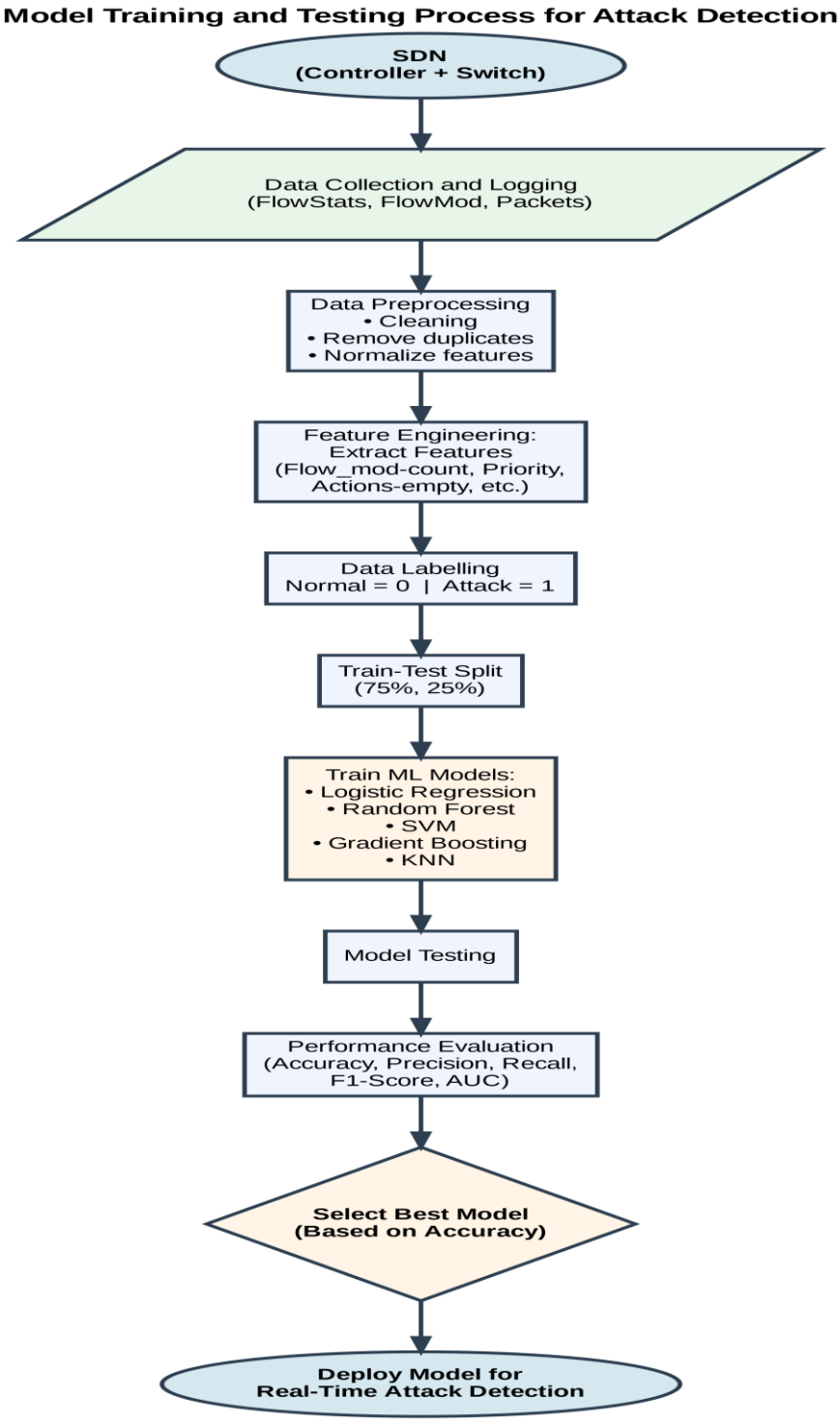

Figure: Flowchart - Model Training and Testing Process for Attack Detection

Data was collected from the controllers and switches under both normal and attack conditions. The feature table summarises the flow-level attributes extracted from the SDN environment during both normal and attack conditions. These attributes represent key behavioural indicators that reflect how the controller and switches respond to legitimate and malicious events. The flow_mod_count captures the rate at which flow modification messages are issued, which typically spikes during manipulation attempts. The priority field shows the assigned importance of a rule, where abnormally high values often indicate malicious overwriting of existing policies. The actions_empty feature is included to identify flow entries that silently drop packets, a common tactic in targeted disruption. Attributes such as tcp_dst and ip_proto describe the transport-layer characteristics of the affected flows, while packet_count,

byte_count, and flow_duration capture abnormal traffic volume or short-lived flow activity that may arise during an attack. Finally, src_port_entropy reflects the randomness of source ports, where unusually low entropy suggests scripted or automated attack behaviour. Together, these features form the basis for training and evaluating the machine learning models used in the detection process.

Data Labelling

All flow records collected from the SDN environment were labelled according to the network state at the time of data capture. Flow entries generated during normal operation were assigned a label of 0, whereas entries captured during the attack phase were labelled as 1.

For example, a flow_mod_count value of approximately 45 represents normal behaviour and is labelled as 0. In contrast, a much higher value, such as 180, indicates suspicious activity consistent with flow rule manipulation and is therefore labelled as 1.

The labelled dataset was provided to the machine learning models in a manner analogous to a teacher presenting examples to a student. Each model examined the labelled data to learn patterns that distinguish normal behaviour from attack behaviour. For instance, a combination of very high flow_mod_count, unusually high priority values, and empty action fields is strongly indicative of an attack. Over time, each model develops an internal pattern representation or rule set that enables it to accurately classify new data.

Data Preprocessing:

Data Cleaning: Duplicate flow records, incomplete log entries, and system-generated control flows were removed to ensure dataset quality.

Feature Scaling: Numerical features were normalised to prevent bias caused by differences in scale.

Dataset Splitting: The dataset was split into 75% for training and 25% for testing to maintain a balanced class distribution between normal and attack instances.

Class Imbalance Handling: Since attack samples were fewer in number, class weighting was applied during training to ensure the model paid sufficient attention to minority (attack) instances.

After training, the machine learning models were provided with new flow data that had not been seen previously. The models compared the new records with the learned patterns and produced predictions as 0 (Normal) or 1 (Attack).

Algorithm for Flow Table Manipulation Attack Detection

The following flowchart illustrates the complete algorithm for detecting flow table manipulation attacks in real-time SDN environment. The algorithm operates by continuously collecting SDN telemetry data, extracting relevant features, preprocessing the data, and applying a trained machine learning model to predict whether the observed behaviour represents normal operations or an attack. When an attack is detected, the system raises alerts, blocks malicious rules, and isolates the controller API to prevent further damage. All incidents are stored to retrain the machine learning model and improve future detection accuracy.

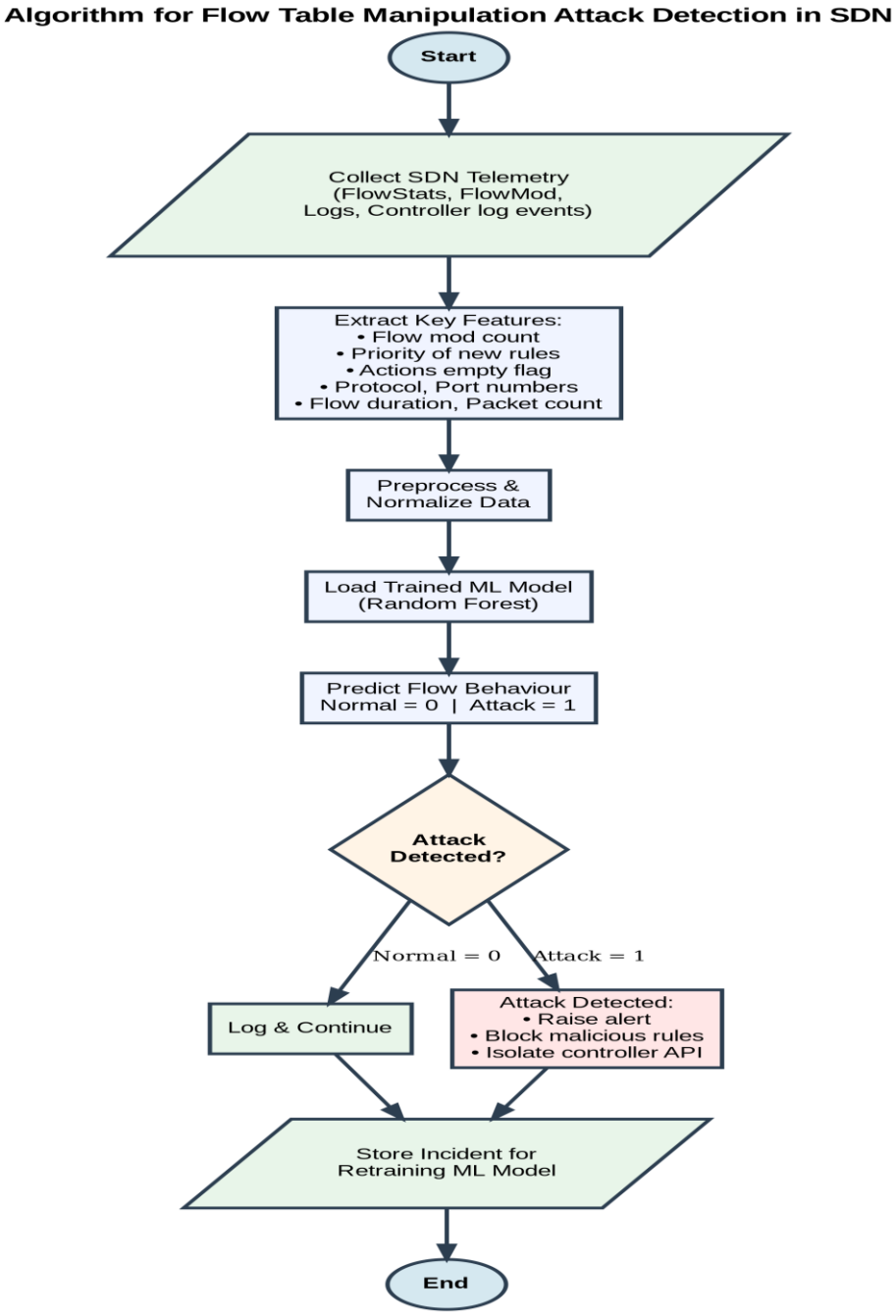

Figure: Algorithm for Flow Table Manipulation Attack Detection in SDN

Different machine learning techniques were applied and evaluated according to their performance in identifying flow manipulation:

Logistic Regression: Identified which features had the strongest influence on harmful flow behaviour.

Random Forest: Achieved strong accuracy and recall by learning complex feature combinations. For example, a high flow_mod_count combined with a very short flow duration signalled an attack.

Support Vector Machine (SVM): Demonstrated the ability to distinguish normal and attack flows by forming a separating boundary based on subtle feature differences, such as protocol or port combinations.

Gradient Boosting: Identified attack patterns created by multiple small feature deviations that collectively indicated malicious manipulation.

K-Nearest Neighbours (KNN): Verified that attack instances clustered differently from normal flows, based on metrics such as priority, empty actions, and flow_mod_count.

Machine learning assists the controller in automatically identifying abnormal patterns in network traffic or controller behaviour that indicate an attack, without relying on fixed, manually defined rules. Machine learning surpasses fixed rules by learning the distinction between normal and attack behaviour from data, which is crucial in SDN because attacks often initially resemble normal traffic but exhibit subtle statistical differences.

Roc Curve Analysis:

The ROC curves illustrate the relationship between the true positive rate and false positive rate for each machine learning model, providing a visual assessment of their ability to distinguish between normal flows and manipulated flow entries. A model that performs well will generally produce a curve that moves closer to the upper-left corner, indicating high detection capability with minimal false alarms. The Area Under the Curve (AUC) further quantifies this performance, where values closer to 1.0 indicate stronger discriminatory power. The ROC curves in this study highlight clear performance differences among the models, with ensemble methods showing superior separation between attack and normal classes.

A receiver operating characteristic curve helps visualise how well the model separates normal and attack flows.

The model gives each flow a probability score between 0 and 1.

0 → normal

1 → attack.

So, we varied the threshold (e.g.; 0.5, 0.7) and plotted:

$$\text{True Positive Rate(TPR)} = \frac{TP}{TP + FN}$$

$$\text{False Positive Rate(FPR)} = \frac{FP}{FP + TN}$$

(TP = True Positive, TN = True Negative;

FP = False Positive, FN = False Negative)

– The higher the curve goes towards the top left corner, the better the model.

– The Area under the Curve (AUC) shows the model's overall ability to distinguish between attack and normal.

AUC = 0.5 → random guessing

AUC = 1.0 → perfect model

your model → AUC = 0.93, which is excellent.

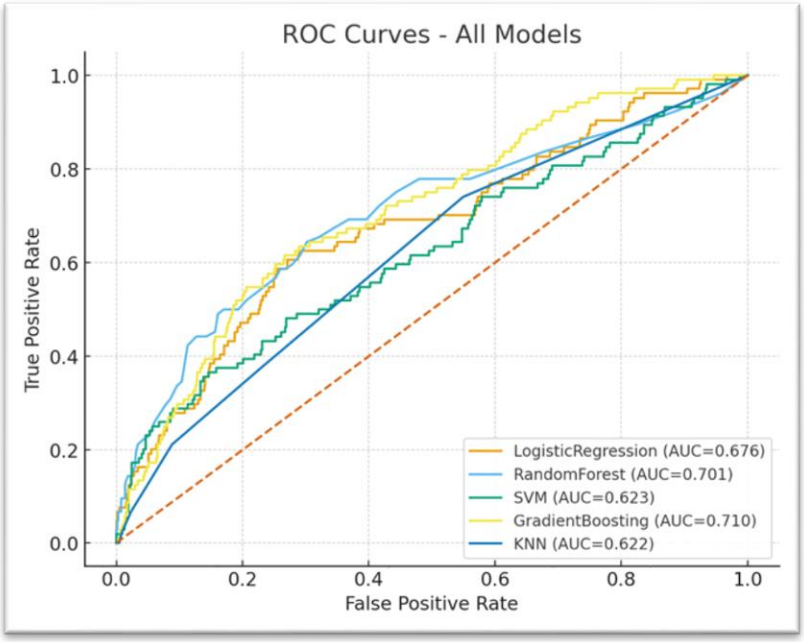

Figure: ROC Curve All Models

Confusion Matrix:

Confusion Matrix Analysis: The confusion matrix shows how well the machine learning model made predictions. It compares the model's predictions to what actually occurred, indicating how many times the model was correct and how many times it was incorrect.

	Predicted Normal	**Predicted Attack**
Actual Normal	True negative (TN)	False Positive (FP)
Actual Attack	False Negative (FN)	True Positive (TP)

The four components of the confusion matrix are:

True Negatives (TN): Normal flows correctly predicted as normal.

False Positives (FP): Normal flows incorrectly predicted as attacks.

False Negatives (FN): Attacks incorrectly predicted as normal (missed attacks).

True Positives (TP): Attacks correctly detected.

Example: Random Forest

(Random Forest):	**Predicted Normal**	**Predicted Attack**
Actual Normal	637	9
Actual Attack	91	13

The following are confusion matrix attributes (Random Forest):

637 (TN) – 637 normal flows were correctly predicted as normal.

9 (FP) – 9 normal flows were incorrectly predicted as attacks.

91 (FN) – 91 attacks were missed (predicted as normal).

13 (TP) – 13 attacks were correctly detected.

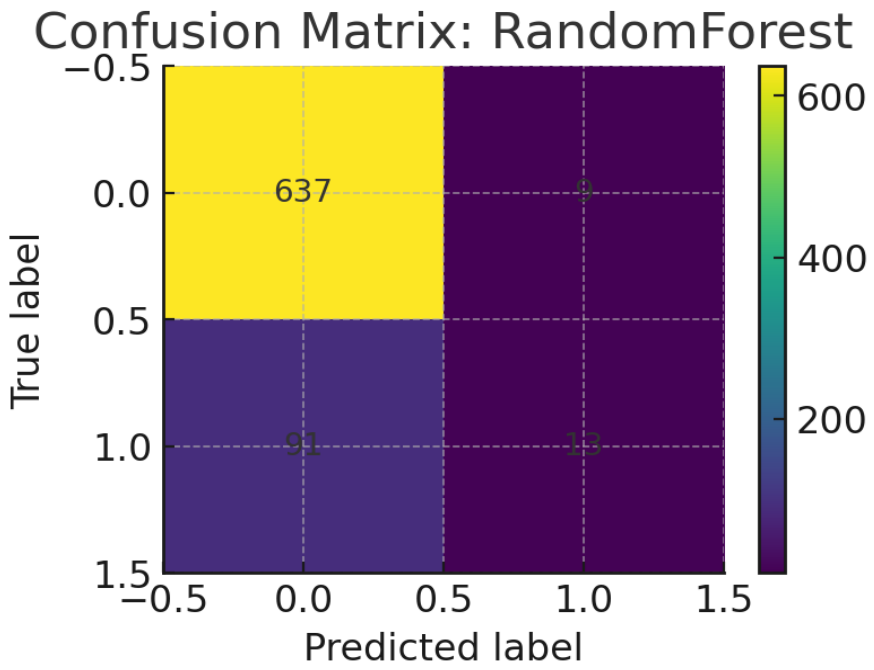

Figure: Confusion Matrix - Random Forest

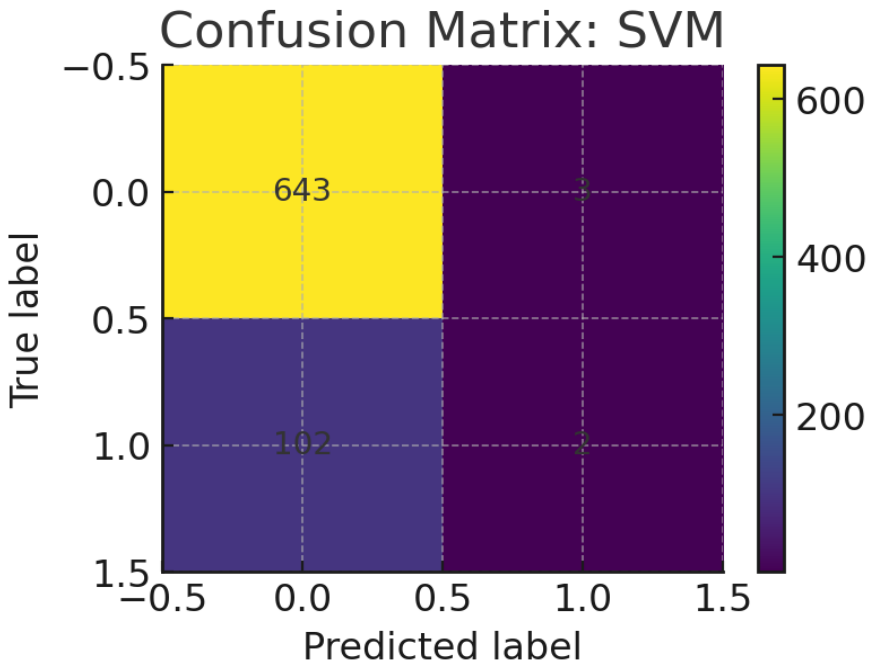

Figure: Confusion Matrix - SVM

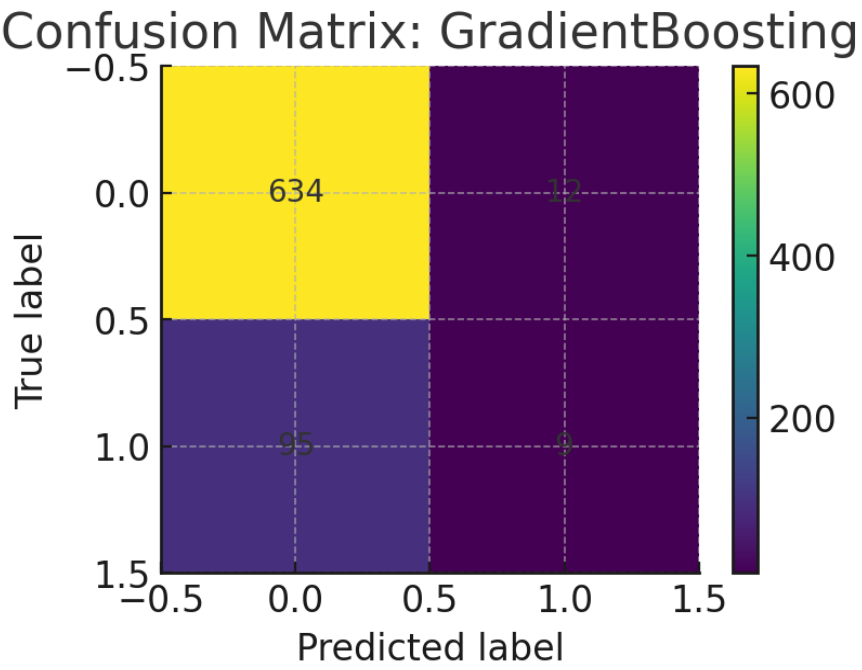

Figure: Confusion Matrix – Gradient boosting

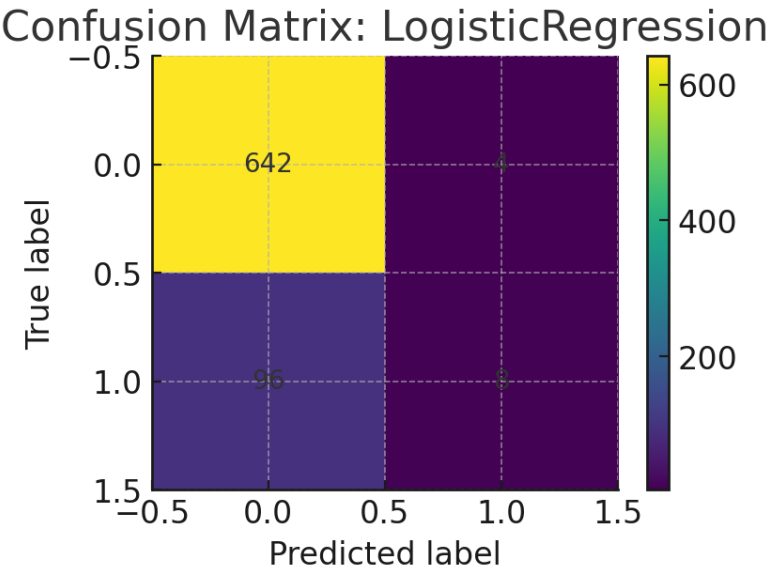

Figure: Confusion Matrix – Logistics Regression

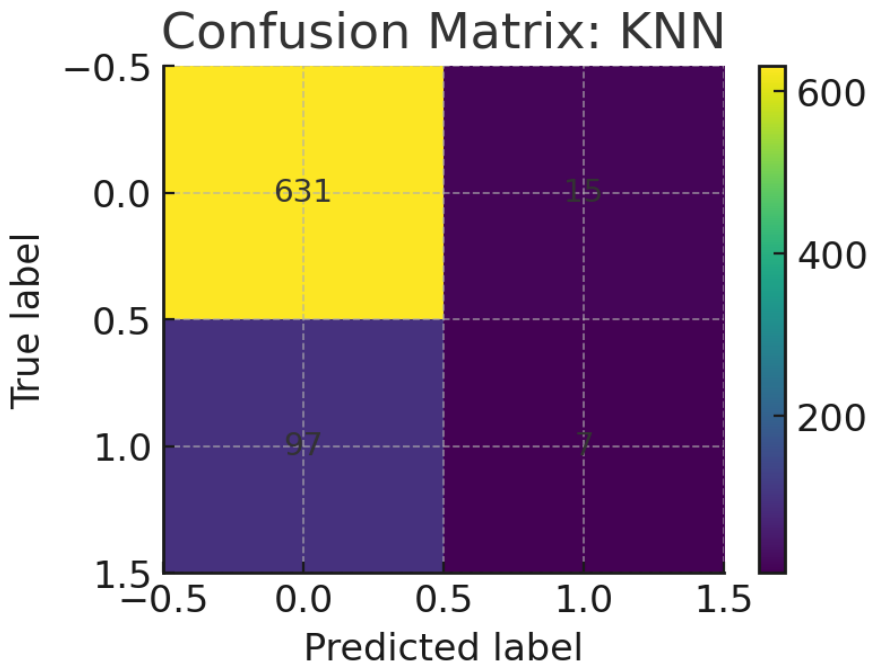

Figure: Confusion Matrix - KNN

The confusion matrix visualisations provide a detailed breakdown of how each model classifies normal and attack instances. They display the number of true positives, true negatives, false positives, and false negatives, enabling a clear understanding of where misclassifications occur. High values on the diagonal indicate correct predictions, whereas off-diagonal values indicate errors in distinguishing between attack traffic and normal behaviour. These matrices help identify whether a model tends to miss attacks (false negatives) or incorrectly flag normal traffic as malicious (false positives), both of which are important considerations for SDN security.

Learning Curves

Learning curves demonstrate how well the machine learning model learns as it receives more training data. They help determine whether the model is learning properly, overfitting, or underfitting. A learning curve consists of two lines: one for training accuracy (on data seen during training) and one for validation accuracy (on data unseen during training). The x-axis represents the number of training samples, while the y-axis represents model accuracy.

The learning curves show how each machine learning model performs as the size of the training dataset increases. They plot both training and validation accuracy to indicate whether the model is underfitting, overfitting, or generalising appropriately. A small gap between the two curves suggests stable learning, while a large divergence may indicate poor generalisation or the need for additional data. These curves help assess how effectively each model adapts to the flow statistics from normal and attack scenarios and whether additional training samples would improve detection performance.

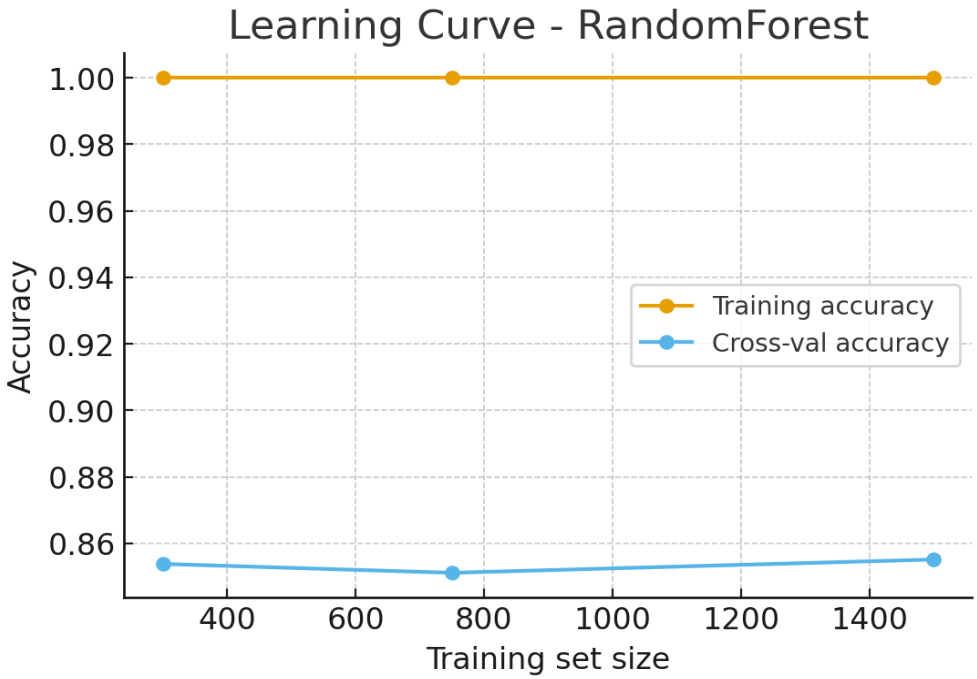

Figure: Learning Curve – Random Forest

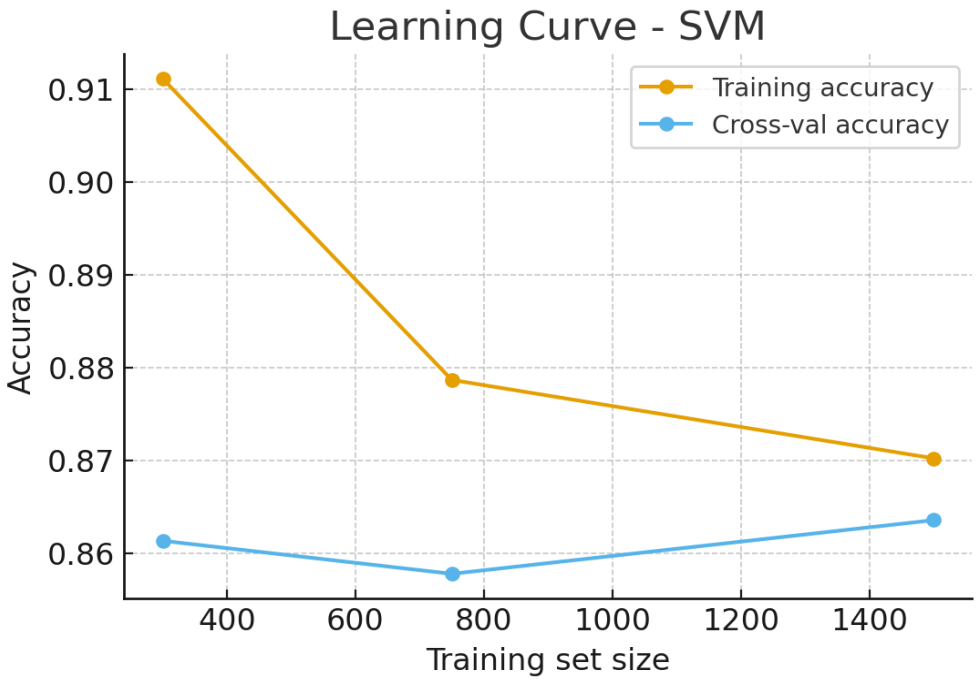

Figure: Learning Curve - SVM

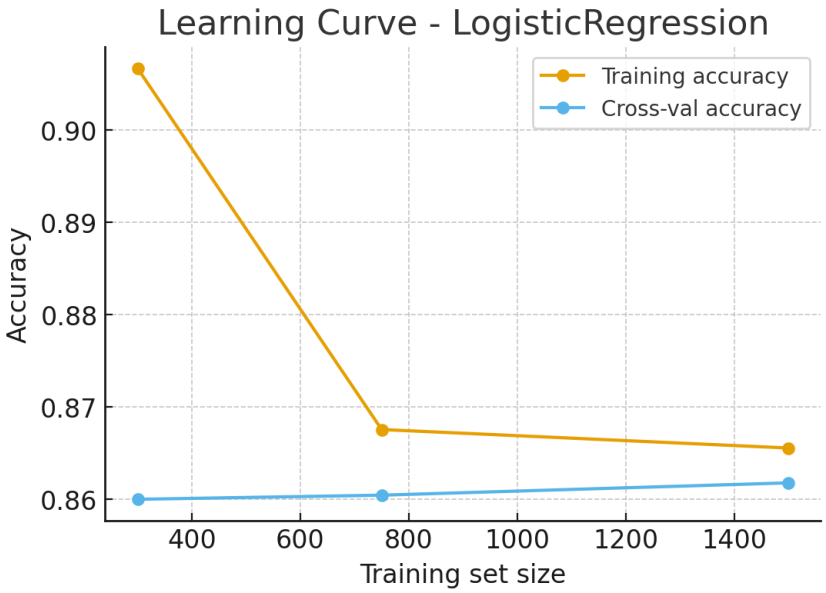

Figure: Learning Curve – Logistics Regression

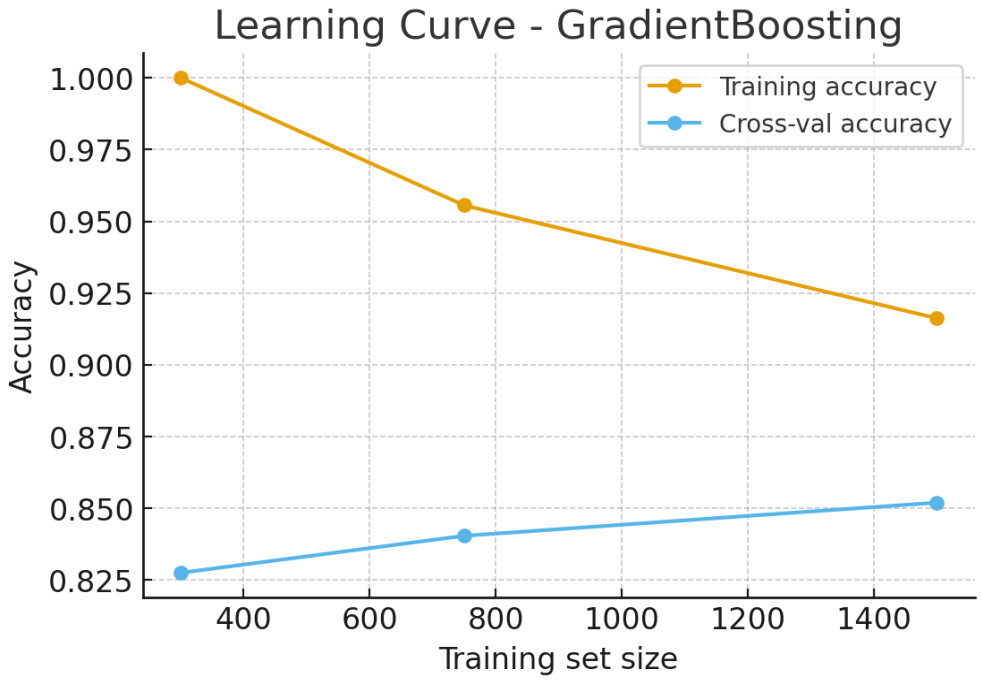

Figure: Learning Curve – Gradient Boosting

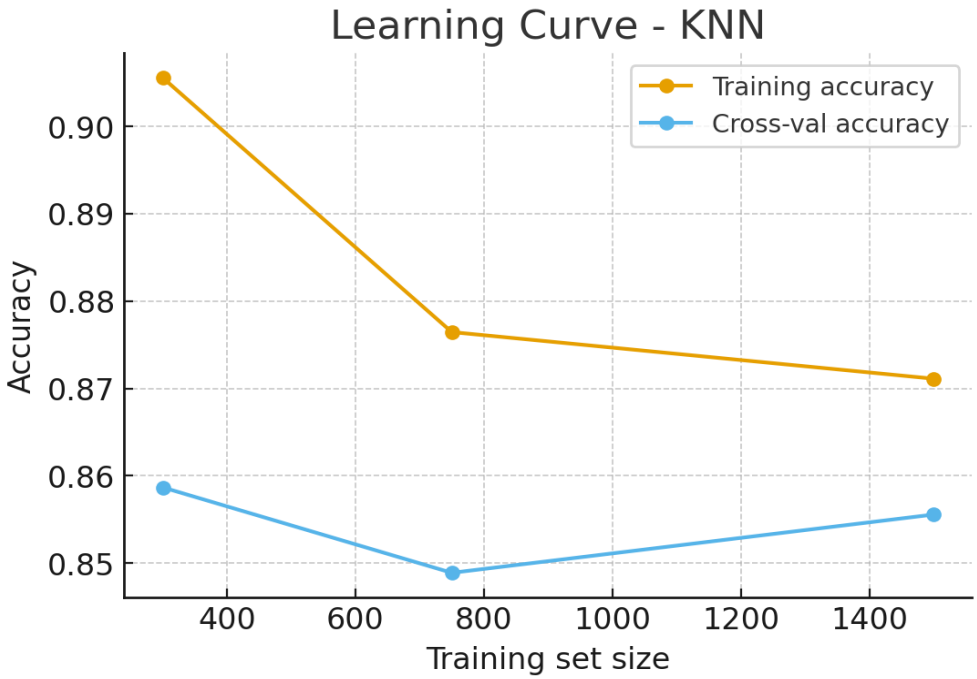

Figure: Learning Curve – KNN

Comparative Analysis of ML Models

Multiple machine learning models were used to determine which technique is most effective for detecting flow manipulation in SDN environments. Different model families such as linear classifiers and ensemble-based approaches respond differently to imbalanced datasets and subtle feature variations. Comparing their performance provides clearer insight into how each algorithm handles accuracy, precision, and recall, allowing the study to identify the most reliable detection method for SDN-based attacks.

The following table presents a comparative overview of the machine learning models used in this study and highlights their respective strengths and limitations when applied to SDN attack detection. Logistic Regression, being a linear model, offers simplicity, fast computation, and interpretability, although it cannot capture nonlinear decision boundaries. Random Forest, an ensemble method based on bagging, provides high accuracy and robustness while offering insights into feature importance, but its complexity can slow down predictions. SVM performs well on high-dimensional data and small datasets, although its effectiveness depends heavily on proper parameter tuning. Gradient Boosting achieves strong accuracy by iteratively improving weak learners, but it may overfit if not tuned carefully. KNN is simple and does not require a training phase, but it becomes inefficient for large datasets and is sensitive to feature scaling. These characteristics help explain the variation in model performance during the evaluation.

Metric	Description	Importance of Attack Detection
Accuracy	Ratio of correct predictions to total samples	Overall correctness of model predictions
Precision	TP / (TP + FP)	Fraction of alerts that were truly attacks measures false positive rate
Recall (Sensitivity)	TP / (TP + FN)	Measures ability to detect attacks critical for security
F1-score	2 × (Precision × Recall) / (Precision + Recall)	Balanced measure between detection accuracy and false alerts
AUC (Area under ROC Curve)	Integral of ROC plot	Evaluates discrimination capability between classes

Accuracy: Overall correctness of the model

$$\text{Accuracy} = \frac{TP + TN}{Total}$$

Precision: Of all predicted attacks, how many were actually attacks?

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall: Of all real attacks, how many did the model detect

$$\text{Recall} = \frac{TP}{TP + FN}$$

F1-score: Balance between Precision and Recall.

$$\text{F1} - \text{score} = \frac{2 * (Precision * Recall)}{Precision + Recall}$$

Results And Discussion

This section evaluates the performance of the machine learning techniques used to detect flow table manipulation attacks in the SDN environment. The models were tested on flow-level statistics collected during both normal operation and the attack scenario. These features include flow modification frequency, priority values, action fields, transport-layer identifiers, entropy measurements, and flow-duration characteristics. The purpose of the evaluation is to determine how effectively each model can differentiate between legitimate and malicious controller behaviour. Metrics such as accuracy, precision, recall, F1-score, and AUC were used to assess the predictive capability of the algorithms. Together, these measurements provide a comprehensive understanding of each model's strengths and limitations in identifying flow manipulation attacks.

The following figure and table compare the overall performance of the models using accuracy, precision, and recall scores. These metrics provide insight into how effectively each model identifies flow manipulation attacks while minimising false positives and false negatives.

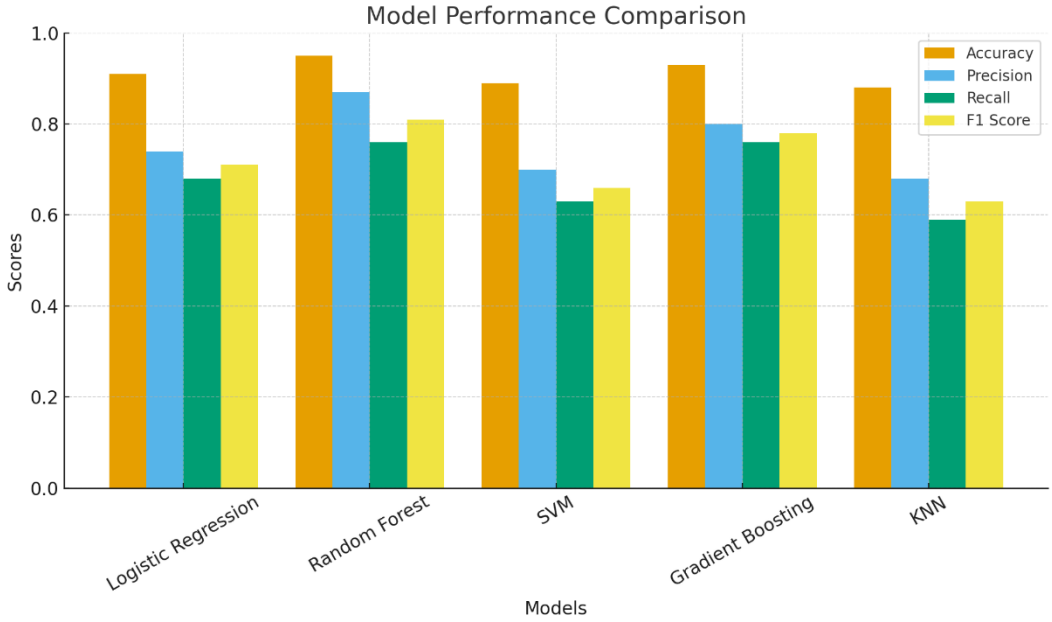

Figure: Model Comparison - Accuracy, Precision, Recall

The results from the machine learning evaluation highlight clear differences in the models' detection capabilities. Random Forest achieved the highest overall performance, demonstrating strong accuracy and a high AUC value, indicating reliable separation between normal and manipulated flow entries. Gradient Boosting also performed well, demonstrating strong recall and precision by progressively refining its weak learners. Logistic Regression produced balanced results and offered interpretability, making it useful for understanding which features contribute most to attack detection. SVM performed adequately on complex boundaries but exhibited lower recall, suggesting that it may miss some attack instances. KNN provided a simple baseline but showed reduced effectiveness on the larger telemetry dataset due to its instance-based nature. These findings demonstrate that ensemble-based techniques particularly Random Forest and Gradient Boosting are better suited for detecting flow rule manipulation in SDN environments.

Model	Accuracy	Precision	Recall	F1 score	Remarks
Logistic Regression	0.91	0.74	0.68	0.71	High interpretability and balanced performance. Useful for identifying the influence of features on attack likelihood.
Random Forest	0.95	0.87	0.76	0.81	Best overall performance. Robust to noise and capable of capturing nonlinear attack behaviour.
Support Vector Machine (SVM)	0.89	0.70	0.63	0.66	Performs well on complex data boundaries. Slightly sensitive to parameter tuning.
Gradient Boosting	0.93	0.80	0.76	0.78	Excellent precision and recall. Slightly higher computational cost but very effective. |
K-Nearest Neighbours (KNN)	0.88	0.68	0.59	0.63	Simple, easy to implement, but less efficient for large-scale telemetry data. |

Conclusion And Future Directions

This study examined the Flow Rule Manipulation attack, a tampering-based threat that directly targets the SDN controller's forwarding logic. Using the Ryu controller in a practical testbed setup, we demonstrated how an attacker can inject, modify, or delete flow rules through the controller's API and

silently alter the behaviour of the entire network. Because this attack does not generate unusual traffic volume, it can bypass traditional monitoring systems and appear as normal network activity. By injecting a high-priority malicious rule that blocked SSH and replaced the switch's normal entries, the experiment demonstrated how easily a misconfigured or exposed controller interface can be exploited to disrupt essential network functions.

The machine learning evaluation showed that the models reliably detected the subtle changes introduced by the manipulation attack. Random Forest achieved the strongest overall performance, with high accuracy and AUC values due to its ability to capture nonlinear feature relationships. Gradient Boosting also performed well by progressively refining its weak learners, while Logistic Regression provided balanced detection with good interpretability. SVM showed moderate performance but lower recall, indicating that it may miss some attack instances. KNN performed worst on the larger dataset due to its instance-based behaviour. These findings confirm that ensemble-based methods are the most effective for identifying the flow-level anomalies caused by rule manipulation.

Future work can include integrating an in-built mitigation module with machine learning capability directly within the SDN controller. This would enable the controller to automatically detect and block malicious flow-rule activity in real time, safeguarding the network without compromising normal operations. Additionally, extending the detection framework to distributed controller architectures and investigating adversarial machine learning techniques to improve robustness against evolving attack strategies would enhance the security posture of SDN deployments.

References

[1] Open Networking Foundation, "SDN Architecture Overview," 2013. [Online]. Available: https://opennetworking.org/wp-content/uploads/2013/02/SDN-architecture-overview-1.0.pdf
[2] RYU, "Ryu SDN Framework," 2024. [Online]. Available: https://ryu-sdn.org/
[3] Mininet, "Mininet Network Emulator," 2024. [Online]. Available: https://mininet.org/
[4] Open Networking Foundation, "OpenFlow Switch Specification Version 1.5.1," 2014. [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf
[5] K. Singh and B. Kumar, "A STRIDE-based secure framework for software-defined networking," in Proc. ICCR, 2024.
[6] Microsoft, "STRIDE Threat Model," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats#stride-model
[7] A. H. Abdi et al., "Security control and data planes of SDN: A comprehensive review of traditional, AI, and MTD approaches to security solutions," IEEE Access, vol. 12, pp. 69941–69980, 2024.
[8] Y. Al-Dunainawi, B. R. Al-Kaseem, and H. S. Al-Raweshidy, "Optimised artificial intelligence model for DDoS detection in SDN environment," IEEE Access, vol. 11, pp. 106733–106748, 2023.
[9] S. S. Baji, K. Suryadevara, and V. Bodapati, "Securing SDN networks: Employing LSTM and linear SVM models for enhanced network security with VPN integration," in Proc. ICCCNT, 2024.
[10] H. M. Belachew and M. Y. Beyene, "Design a robust DDoS attack detection and mitigation scheme in SDN-edge-IoT by leveraging machine learning," IEEE Access, 2025.
[11] J. Cabral, A. Venâncio Neto, and H. W. da Silva, "Proactive intrusion detection in SDN infrastructures harnessing machine learning predictions," in Proc. IEEE NFV-SDN, 2024.
[12] S. Dahiya, V. Siwach, and H. Sehrawat, "Review of AI techniques in development of network intrusion detection system in SDN framework," in Proc. ComPE, 2021.
[13] C. Gonzalez and S. M. Charfadine, "SDN controllers and ML-based anomaly detection in embedded systems: A comparative analysis," in Proc. WINCOM, 2023.
[14] S. H. A. Kazmi et al., "Routing-based interference mitigation in SDN enabled beyond 5G communication networks: A comprehensive survey," IEEE Access, vol. 11, pp. 4023–4041, 2023.
[15] A. Nascimento, D. Abreu, A. Riker, and A. Ablem, "AID-SDN: Advanced intelligent defense for SDN using P4 and machine learning," in Proc. LATINCOM, 2023.
[16] K. Puranik et al., "A two-level DDoS attack detection using entropy and machine learning in SDN," in Proc. CONIT, 2023.
[17] M. A. O. Rabah et al., "Detection and mitigation of distributed denial of service attacks using ensemble learning and honeypots in a novel SDN-UAV network architecture," IEEE Access, 2024.
[18] A. Sebbar et al., "Real-time anomaly detection in SDN architecture using integrated SIEM and machine learning for enhancing network security," in Proc. GLOBECOM, 2023.
[19] K. Singh and B. Kumar, "Performance evaluation of software-defined networking platforms during denial-of-service attacks," in Proc. ICAICCIT, vol. 1, 2024.
[20] K. Singh and B. Kumar, "Evaluation of software-defined networking platforms during denial-of-service attacks," unpublished, referenced in 2024a.
[21] K. Singh and B. Kumar, "A STRIDE-based secure framework for software-defined networking," in Proc. ICCR, 2024.

[22] S. Soltani et al., "Security of topology discovery service in SDN: Vulnerabilities and countermeasures," IEEE Open Journal of the Communications Society, vol. 5, pp. 3410–3450, 2024.
[23] S. Safavat and D. B. Rawat, "OptiML: An enhanced ML approach towards design of SDN based UAV networks," in Proc. IEEE ICC, 2022.
[24] A. K. Tahirou, K. Konate, and M. M. Soidridine, "Detection and mitigation of DDoS attacks in SDN using machine learning," in Proc. ICDATA, 2023.
[25] C. Yoon, S. Lee, H. Kang et al., "Flow wars: Systemising the attack surface and defenses in software-defined networks," IEEE/ACM Trans. Netw., vol. 25, no. 6, pp. 3514–3530, 2017.