# MULTILEVEL INTRUSION DETECTION WITH LOG MANAGEMENT IN CLOUD COMPUTING

**Received**
**April 15, 2024**

**Revised**
**July 16, 2024**

**Accepted**
**August 01, 2024**

Ghaniyyat Bolanle Balogun[1]; Clinton Ifeoluwa Ibitoye[1], Maryam Mero Woru[1]
Fatima Usman-Hamza[1], Salihu Shakirat aderonke[1], Olumuyiwa James Peter[2]
[1]*Department of Computer Sciences, University of Ilorin, Nigeria*
[2]*Department of Mathematical and Computer Sciences, University of Medical Sciences Ondo, Nigeria*
*e-mail: peterjames4real@gmail.com*

**Abstract: Cloud computing is the on-demand availability of computer system resources, especially data storage (cloud storage) and computing power, without direct active management by the user. It is an Information Technology (IT) model that provides on-demand hardware and software services to customers. However, cloud computing systems are vulnerable to various cyber-attacks, often due to poor cybersecurity management or misconfigured services. Therefore, these systems must include Intrusion Detection Systems (IDSs) to safeguard each of their Virtual Machines (VMs) against attacks. Noteworthy is the trade-off between the security level of IDSs and system performance. If the IDS delivers greater security service by employing more rules or patterns, it will require more computer resources in proportion to the level of protection, thereby reducing resources allocated to consumers. Additionally, the large volume of logs in cloud computing may be difficult for system administrators to analyze.**

**In this paper, we introduce a Multi-Level Intrusion Detection System with Log Management for Cloud Computing. This system is implemented on a hypervisor virtual machine (VM) and its efficiency is tested by comparing the algorithm with other existing algorithms. We employ a Machine Learning approach to study various patterns of intrusion using the KDD CUP'99 dataset. The proposed architecture is successfully implemented with Artificial Neural Network (ANN) model training and the integration of the Adaptive Fuzzy C-Means (AFCM) clustering algorithm. Key findings include a significant improvement in detecting intrusions while maintaining optimal resource allocation and system performance. This approach provides a robust solution for Cloud Computing systems to achieve both effective resource utilization and strong security services without compromising either.**

*Keywords: Cloud computing; Machine Learning; Multi Level Intrusion Detection System; Adaptive Fuzzy C Means*

## 1. INTRODUCTION

Cloud computing has evolved into an essential component of modern IT infrastructure, allowing for on-demand access to shared resources such as storage, computing power, and network services. Cloud Computing has recently received more attention than traditional computer services due to its ability to provide an infinite amount of resources. Furthermore, clients can access the services from anywhere that has internet access, making Cloud Computing a good choice in terms of accessibility. Cloud computing allows for resource sharing in the form of scalable infrastructures, middleware and application development platforms, and commercial applications with added value. Some fundamental characteristics of this collection of resources include on-demand self-service, broad network access, multi-tenancy, and ease of maintenance [1-5]

Due to the volume of sensitive data and resources, cloud computing systems are easily targeted by attackers. System administrators, in particular, are vulnerable to becoming attackers. As a result, cloud computing service providers must protect their systems from both internal and external threats. As cloud infrastructures become more complex and large, they become vulnerable to a variety of cyber threats, including intrusion attacks. Cloud computing is vulnerable to a variety of security risks, both intentional and unintentional [6-7]. Threats to the integrity, confidentiality, and availability of cloud resources, data, and infrastructure are examples of such risks. Using a cloud with significant processing and storage capacity for malicious purposes can turn the cloud into a threat to society. Intentional threats can come from both insiders and outsiders. Insiders are legitimate cloud users who use their credentials to gain unauthorized access to the cloud. An intrusion is a type of attack that takes advantage of a security flaw and violates the system's security policy [8]. Despite the fact that a breach implies a successful attack, intrusion detection systems are also designed to detect attempts that do not result in breaches.

Intrusion detection systems (IDSs) have evolved into a critical defense mechanism against these attacks, allowing for the detection of suspicious activity and the prevention of malicious behavior. Multi-level intrusion detection systems (MIDSs) are a type of advanced intrusion detection system (IDS) that can provide enhanced protection for cloud environments. Intrusion Detection System is one of the most often used method for defending Cloud Computing systems against many forms of attacks. An IDS can handle Cloud Computing on a worldwide scale since it monitors traffic from each VM and creates alarm logs. Another significant issue is log management. Because cloud computing systems are utilized by so many individuals, they create a massive number of logs [9-15].

As an integral component of the computing environment, the underlying network infrastructure of a cloud may be attacked. Grid and cloud apps that operate on vulnerable hosts are likewise a security risk. Assaults on any network or host participating in a cloud are considered attacks on that cloud since they may directly or indirectly damage its security characteristics. Because of its innovative protocols and services, cloud systems are vulnerable to all common network and computer security assaults, as well as unique techniques of attack [16].

An intrusion detection system (IDS) is a software or hardware-based solution that monitors network traffic or system events to detect potential security breaches. IDS systems are designed to analyze network traffic and system logs to identify patterns of activity that indicate potential security threats. IDSs are a popular type of security technology. When an IDS identifies a signature of an accident in accordance with host or network security rules, it notifies system administrators and generates an attack log. IDS can be placed in either a host or a network, depending on the objective. Thus, the goal of the IDS is to warn or tell the system that harmful actions have occurred and to attempt to eradicate them.

The detection findings can be reported in three ways: notification, manual response, and automatic response. Intrusion detection systems use various techniques to detect and analyze network traffic or system events. Signature-based IDS systems use a database of known attack patterns to identify potential threats. These systems compare network traffic or system events with the database of known signatures to detect signs of malicious activity. Anomaly-based IDS systems, on the other hand, use statistical models to identify abnormal network traffic or system events. These systems learn from normal network or system behavior and raise alerts when deviations from the norm are detected [17-18].

A cloud intrusion detection system (IDS) is a security solution designed to detect and respond to unauthorized access and other security threats in cloud computing environments. It operates by monitoring network traffic, system logs, and other sources of data to identify suspicious activity that may indicate a potential security breach. Cloud IDS solutions typically consist of two main components: the detection engine and the response engine. The detection engine analyzes network traffic and system logs to identify patterns of behavior that may indicate a security threat. This can include detecting anomalies in user behavior, network traffic, and system performance. The response engine takes action to mitigate the security threat, which may include blocking network traffic, shutting down systems, or sending alerts to security personnel. The manuscript makes a substantial contribution to academic discourse by introducing a novel multi-level intrusion detection system with integrated log management for cloud computing. By employing machine learning techniques, specifically the ANN model and Adaptive Fuzzy C-Means clustering algorithm, it addresses the critical balance between security and system performance. This approach not only enhances the effectiveness of resource utilization but also maintains robust security measures, thereby offering a valuable foundation for future research in optimizing cloud computing security.

## 2. METHODOLOGY

The system proposed is a hybrid intrusion detection system, the system is divided to three phases. The first phase is data preparation, training of the machine learning algorithms, and the final stage is the system implementation.

In this study, we designed a Multi-Level Intrusion Detection System (IDS) with Log Management for Cloud Computing, using a hypervisor VM for deployment and the KDD CUP'99 dataset for training. We employed Artificial Neural Networks (ANN) and Adaptive Fuzzy C-Means (AFCM) clustering for intrusion detection, and implemented centralized log management using Elasticsearch and Kibana for efficient data handling. Our approach was evaluated on key performance metrics, demonstrating significant improvements in intrusion detection and resource optimization. The methodology ensures reproducibility and provides a robust foundation for future cloud security research.

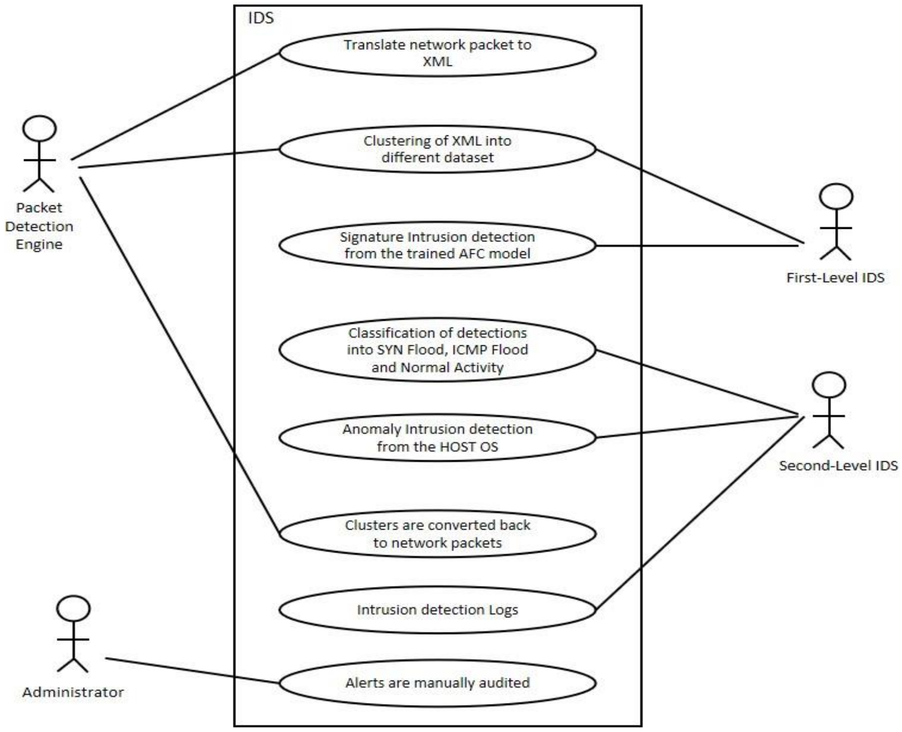



Figure 2.1: Use Case of the proposed system

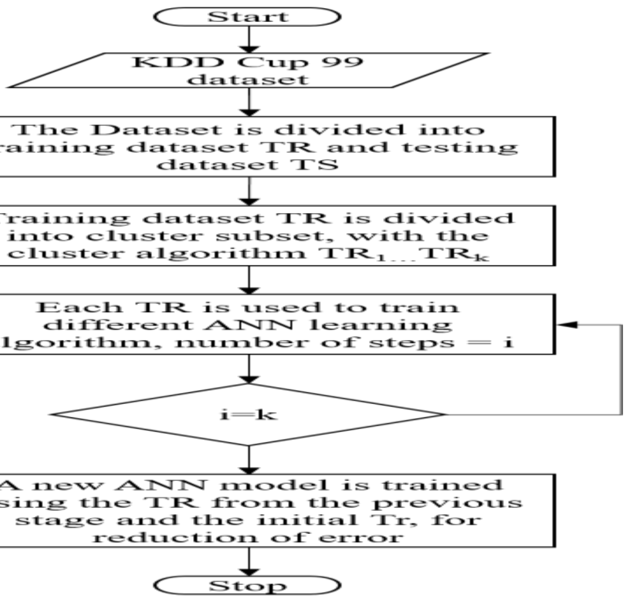



Figure 2.2: Flowchart of the proposed system

Figure 2.1 shows the interaction between the actors;
Packet Detection Engine: The packet detection engine translates incoming network packets from the host OS to the cloud i.e Guest OS, it translates it to the XML format, the detection is done with the BROS python framework, after detection it clusters the data to a suitable form for the proposed Intrusion Detection System.
First-Level IDS: The First Level IDS get the data clusters from the packet detection engine and classify the various data into normal, probe, U2R attack from the trained model.

Second-Level IDS: Second Level IDS performs the anomaly intrusion detection, from the identified normal state of the dataset gotten from the ANN models it check if the packet is at a normal state or it's an intrusion like ICMP flood, it's output is recorded in a log file.

Administrator: The Administrator is responsible for manually monitoring of the logs from the IDS and taking actions on preventing it.
Figure 2.2 shows the overall system flowchart from the point of data preparation to clustering of training subsets by the clustering algorithm, then to training of different ANN model with the dataset.

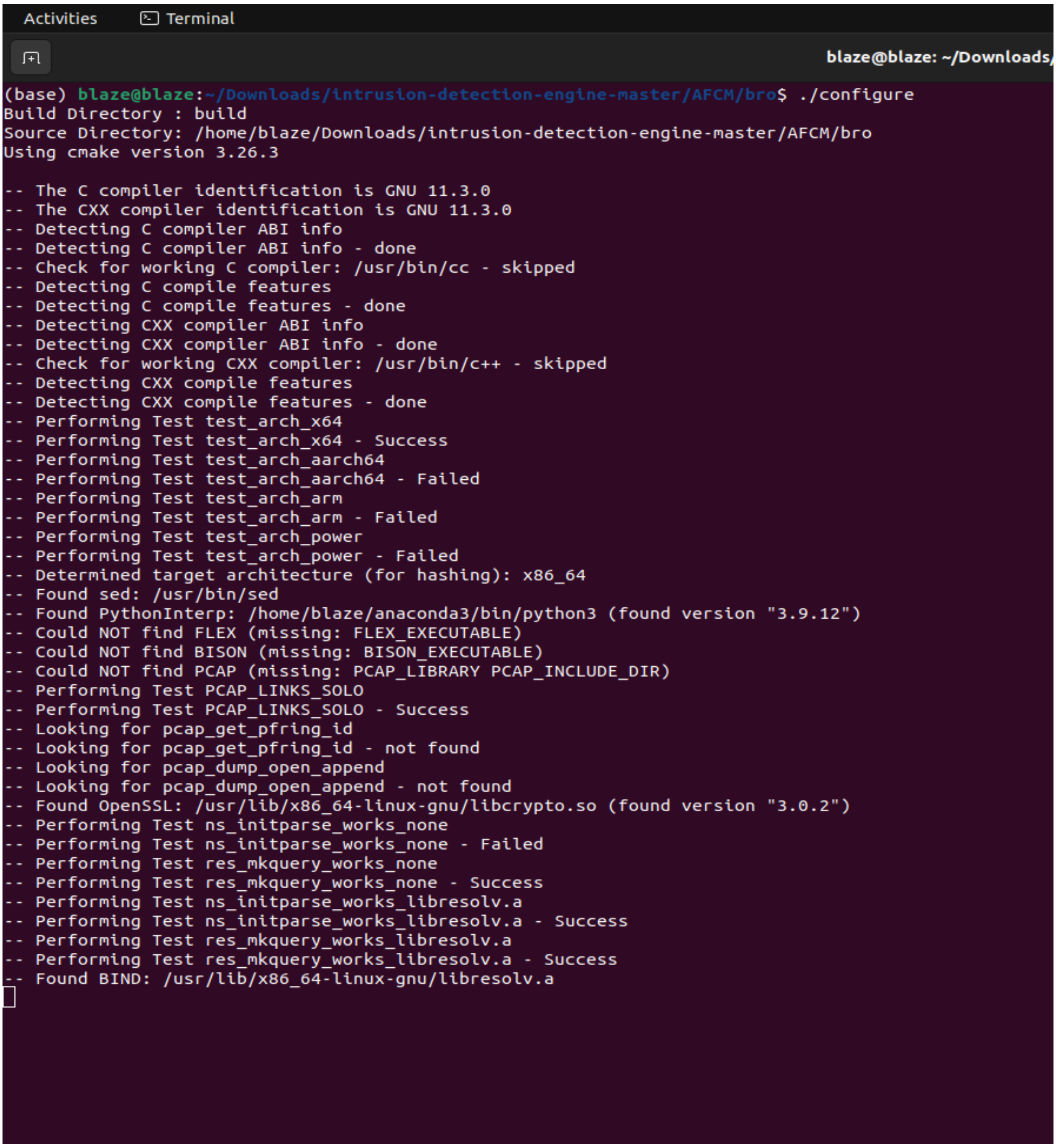


Figure 2.3: Installation of library and dependencies on Ubuntu

The KDD Cup '99 dataset is a widely used benchmark dataset in intrusion detection and network security. It was created by the Defense Advanced Research Projects Agency (DARPA) as part of an effort to improve the security of computer networks. The dataset contains network traffic data captured over a nine-week period from a simulated environment that emulates a typical U.S. Air Force LAN. The dataset has 498000 labeled data points, each point containing 42 attributes pertaining to the basic features of TCP connections, domain related features and certain time based statistics, the dataset are of four categories of intrusion which are denial of service (DoS), probe (PRB), remote to local (R2L) and user to root (U2R).

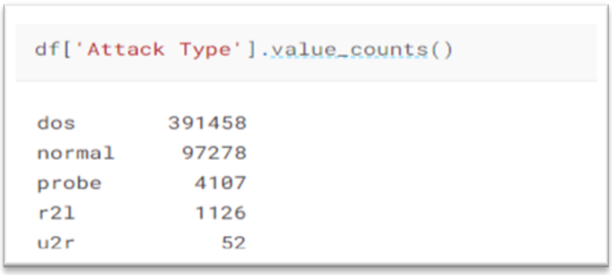



Figure 2.4: Classification of attacks

The dataset was cleaned and used in training the ANN model, duplicate were removed, only 11 categories used from the initial 42 categories of the original dataset. From the modified dataset 4600 data points were selected for training. Table 2.1 shows the 11 attributes used by the training dataset.

| S/N | Attribute | Description | Type |
|---|---|---|---|
| 1 | Duration | Duration of the connection | Continuous |
| 2 | Service | Destination service | Discrete |
| 3 | Src bytes | bytes sent from source to destination | Continuous |
| 4 | Dst bytes | bytes sent from destination to source | Continuous |
| 5 | Count | number of connections to the same host as the current connection in the past two seconds | Continuous |
| 6 | Srv_count | number of connections to the same service as the current connection in the past two seconds | Continuous |
| 7 | Dst_host_count | count of connections having the same destination host | Continuous |
| 8 | Dst_host_srv_count | count of connections having the same destination host and using the same service | Continuous |
| 9 | Dst_host_diff_srv_rate | % of connections to the current host having the same src port | Continuous |
| 10 | Dst_host_same_src_port_rate | % of connections to the current host having the same src port | Continuous |
| 11 | Dst_host_serror_rate | % of connections to the current host that have an S0 error | Continuous |

**Data Preprocessing and Data cleaning**

The Dataset used contain over 4 million data points with various types of attacks, the data need some cleaning and classification. The classification is done based on the types of data in the class column, it's was classified into five classes, normal, DOS, probe, R2L, U2R.


Figure 2.5: Importing of the KDD dataset

After cleaning of the data and reducing datapoint from 4 million to five hundred thousand the new updated dataset is stored in kddcup.data.corrected. It is visualized with pandas library and plotted with the GNUPLOT to get the accuracy of the membership.

```
In [20]: df.info()
         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 500000 entries, 0 to 499999
         Data columns (total 42 columns):
          #   Column   Non-Null Count   Dtype
         ---  ------   --------------   -----
          0   0        500000 non-null  int64
          1   tcp      500000 non-null  object
          2   http     500000 non-null  object
          3   SF       500000 non-null  object
          4   215      500000 non-null  int64
          5   45076    500000 non-null  int64
          6   0.1      500000 non-null  int64
          7   0.2      500000 non-null  int64
          8   0.3      500000 non-null  int64
          9   0.4      500000 non-null  int64
          10  0.5      500000 non-null  int64
          11  1        500000 non-null  int64
          12  0.6      500000 non-null  int64
          13  0.7      500000 non-null  int64
          14  0.8      500000 non-null  int64
          15  0.9      500000 non-null  int64
          16  0.10     500000 non-null  int64
          17  0.11     500000 non-null  int64
          18  0.12     500000 non-null  int64
          19  0.13     500000 non-null  int64
          20  0.14     500000 non-null  int64
          21  0.15     500000 non-null  int64
          22  1.1      500000 non-null  int64
          23  1.2      500000 non-null  int64
          24  0.00     500000 non-null  float64
```

Figure 2.6: Data structure of the dataset

Figure 2.5 shows the data type of the initial dataset with 42 colunms and 500000 datapoint, most of the data type used is the int64. Figure 2.6 is the updated dataset after cleaning and data processing, it contains 12 colunms, the last indes {12} is used to specify the types of attack and services.

```
 40  0.00.15  500000 non-null  float64
 41  normal.  500000 non-null  object
dtypes: float64(15), int64(23), object(4)
memory usage: 160.2+ MB

In [23]: df.drop(df.columns
                 [[1, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 24, 25,
                   26, 27, 28, 29, 30, 33, 36, 38,39,40]], axis=1, inplace=True)

In [24]: df.head()

Out[24]:
```

| | 0 | http | 215 | 45076 | 1.1 | 1.2 | 0.16 | 0.17 | 0.00.7 | 0.00.8 | 0.00.10 | normal. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | http | 162 | 4528 | 2 | 2 | 1 | 1 | 0.0 | 1.00 | 0.0 | normal. |
| **1** | 0 | http | 236 | 1228 | 1 | 1 | 2 | 2 | 0.0 | 0.50 | 0.0 | normal. |
| **2** | 0 | http | 233 | 2032 | 2 | 2 | 3 | 3 | 0.0 | 0.33 | 0.0 | normal. |
| **3** | 0 | http | 239 | 486 | 3 | 3 | 4 | 4 | 0.0 | 0.25 | 0.0 | normal. |
| **4** | 0 | http | 238 | 1282 | 4 | 4 | 5 | 5 | 0.0 | 0.20 | 0.0 | normal. |

```
In [ ]:
```

Figure 2.7: Updated dataset after data cleaning

**Alternative Fuzzy c-mean clustering**

The primary purpose of this step to generate different homogeneous datasets from the heterogeneous training dataset based on fuzzy membership values. The data within the same cluster must have homogeneity and the data belonging to different clusters should have heterogeneity. So, in this phase, the whole training data TR undergoes alternative fuzzy c-means clustering method. The fuzzy membership values for the data points is obtained using given distance function from the AFCM algorithm.

The process of clustering is an iterative process which is performed by calculating membership values, cluster centers at every step and optimization of given cost function.

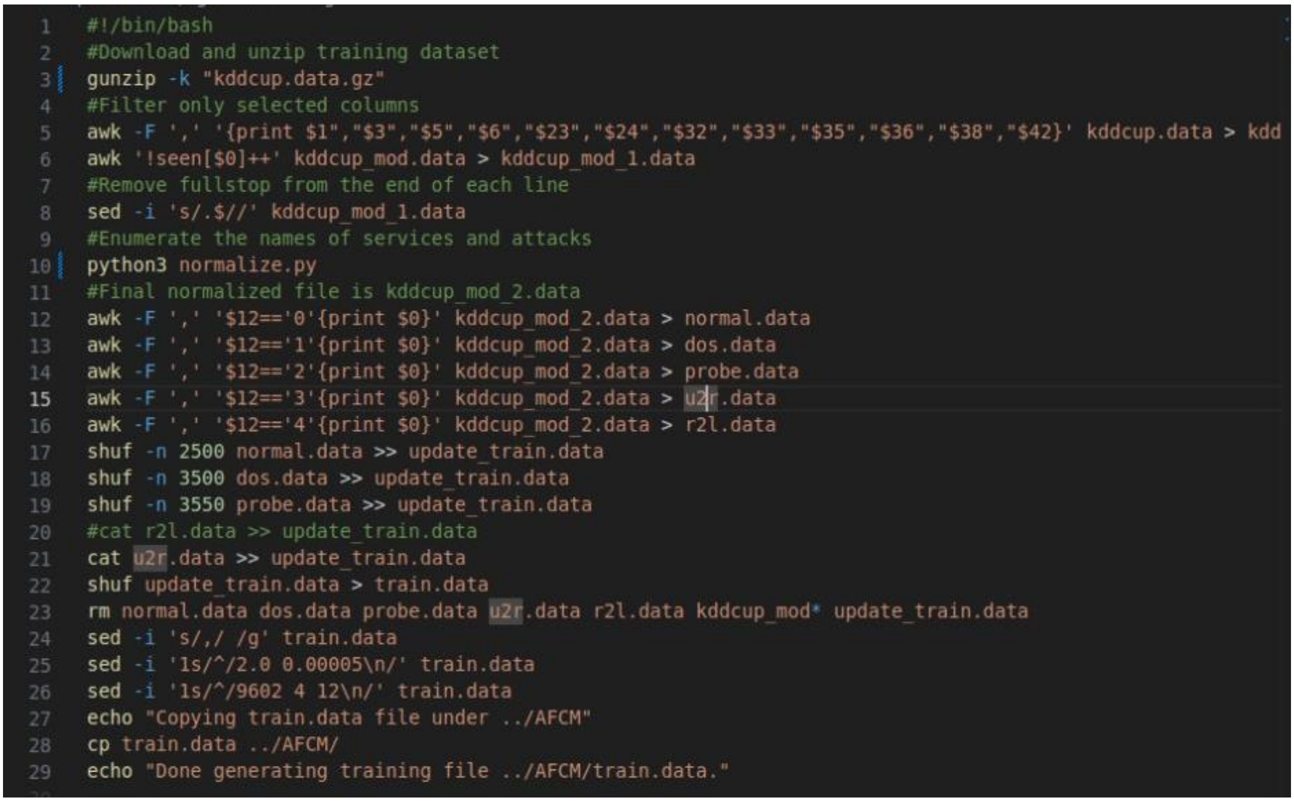
***2.3 Artificial Neural Network (ANN)***

The ANN module is used to learn about the patterns present in the dataset and make decisions on how closely a given attack or normal interaction in the network is related to the given patterns. The module is a feed forward neural network, trained via back-propagation. The module has an input layer, a hidden layer and an output layer. An input to the ANN would be multiplied by a weight and fed to the hidden layer. The ANN are divided into three different stages Stage I, stage II and stage III for optimal neural network. Keras library was used in training the models and the ANN stage input as dependent on the previous stage.

### *2.5 Fuzzy Aggregation*

The three stages of ANN layers are aggregated together for optimal deep learning. After the ANNs of Stage II have been trained with their respective datasets, another ANN is trained with the combined details of the previous ANNs. This ANN has the same number of inputs as the number of outputs of the ANNs in Stage II. The input for the new ANN is formed using matrix multiplication. Every value of the output nodes of stage II are multiplied with the membership values of that point. This gives preference to ANNs which have data fed into them with higher membership values as the training data was created using alternative fuzzy clustering. Log Management Syslog is responsible for the log management, when the IDS detect intrusion in the host it stores a log file with the type of attack and also notify the system administrator to take action.

The KDD Cup 99 dataset is extracted and cleaned with various parameters. It narrows down the attacks to only four types of attacks which are DoS, probe, U2R, R2L. This data gotten after the cleaning is divided to two data set which are training dataset and testing dataset. The process and result of the data preprocessing and cleaning is shown in Figure 2.8 .



```
#!/bin/bash
#Download and unzip training dataset
gunzip -k "kddcup.data.gz"
#Filter only selected columns
awk -F ',' '{print $1","$3","$5","$6","$23","$24","$32","$33","$35","$36","$38","$42}' kddcup.data > kdd
awk '!seen[$0]++' kddcup_mod.data > kddcup_mod_1.data
#Remove fullstop from the end of each line
sed -i 's/.$//' kddcup_mod_1.data
#Enumerate the names of services and attacks
python3 normalize.py
#Final normalized file is kddcup_mod_2.data
awk -F ',' '$12=='0'{print $0}' kddcup_mod_2.data > normal.data
awk -F ',' '$12=='1'{print $0}' kddcup_mod_2.data > dos.data
awk -F ',' '$12=='2'{print $0}' kddcup_mod_2.data > probe.data
awk -F ',' '$12=='3'{print $0}' kddcup_mod_2.data > u2r.data
awk -F ',' '$12=='4'{print $0}' kddcup_mod_2.data > r2l.data
shuf -n 2500 normal.data >> update_train.data
shuf -n 3500 dos.data >> update_train.data
shuf -n 3550 probe.data >> update_train.data
#cat r2l.data >> update_train.data
cat u2r.data >> update_train.data
shuf update_train.data > train.data
rm normal.data dos.data probe.data u2r.data r2l.data kddcup_mod* update_train.data
sed -i 's/,/ /g' train.data
sed -i '1s/^/2.0 0.00005\n/' train.data
sed -i '1s/^/9602 4 12\n/' train.data
echo "Copying train.data file under ../AFCM"
cp train.data ../AFCM/
echo "Done generating training file ../AFCM/train.data."
```

Figure 2.8: Data Cleaning and Processing

The attacks are classified into four categories, and their corresponding classifier number. 1 is a normal attack, 2 is a probe attack, 3 is the User to Root attack and 4 is the Root to Local attack.

Figure 2.9 shows the updated dataset after data cleaning and preprocessing, the services and attack protocols are also shown.

```
 41  normal.   500000 non-null   object
dtypes: float64(15), int64(23), object(4)
memory usage: 160.2+ MB

In [23]: df.drop(df.columns
                [[1, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 24, 25,
                  26, 27, 28, 29, 30, 33, 36, 38,39,40]], axis=1, inplace=True)

In [24]: df.head()
Out[24]:
```

| | 0 | http | 215 | 45076 | 1.1 | 1.2 | 0.16 | 0.17 | 0.00.7 | 0.00.8 | 0.00.10 | normal. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | http | 162 | 4528 | 2 | 2 | 1 | 1 | 0.0 | 1.00 | 0.0 | normal. |
| 1 | 0 | http | 236 | 1228 | 1 | 1 | 2 | 2 | 0.0 | 0.50 | 0.0 | normal. |
| 2 | 0 | http | 233 | 2032 | 2 | 2 | 3 | 3 | 0.0 | 0.33 | 0.0 | normal. |
| 3 | 0 | http | 239 | 486 | 3 | 3 | 4 | 4 | 0.0 | 0.25 | 0.0 | normal. |
| 4 | 0 | http | 238 | 1282 | 4 | 4 | 5 | 5 | 0.0 | 0.20 | 0.0 | normal. |

```
In [ ]:
```

Figure 3.9 Updated dataset from the data preprocessing output

### *2.8. Fuzzy Clustering*

The membership Matrix in the AFCM algorithm represents the degree of membership of each data point to each cluster, and it's obtained through iterative updates based on the distance of data points to cluster centers, considering adaptive weights. Figure 3.0 shows the membership matrix of the data.

```
Open    membership.matrix  ~/ids/intrusion-detection-engine/AFCM    Save
Membership matrix:
Data[0]: 0.001836 0.997706 0.000000 0.000458
Data[1]: 0.000550 0.999307 0.000000 0.000143
Data[2]: 0.207160 0.580879 0.000019 0.211942
Data[3]: 0.161308 0.739523 0.000012 0.099156
Data[4]: 0.000511 0.999356 0.000000 0.000133
Data[5]: 0.001262 0.998440 0.000000 0.000298
Data[6]: 0.000670 0.999155 0.000000 0.000175
Data[7]: 0.005038 0.993652 0.000000 0.001310
Data[8]: 0.000599 0.999245 0.000000 0.000156
Data[9]: 0.000513 0.999353 0.000000 0.000134
Data[10]: 0.000773 0.999025 0.000000 0.000202
Data[11]: 0.000573 0.999278 0.000000 0.000149
Data[12]: 0.000646 0.999185 0.000000 0.000169
Data[13]: 0.000654 0.999175 0.000000 0.000171
Data[14]: 0.890551 0.081624 0.000009 0.027815
Data[15]: 0.000538 0.999321 0.000000 0.000140
Data[16]: 0.168408 0.782097 0.000012 0.049483
Data[17]: 0.000645 0.999187 0.000000 0.000168
Data[18]: 0.000522 0.999342 0.000000 0.000136
Data[19]: 0.000525 0.999339 0.000000 0.000137
Data[20]: 0.000516 0.999349 0.000000 0.000135
Data[21]: 0.000545 0.999313 0.000000 0.000142
Data[22]: 0.000503 0.999366 0.000000 0.000131
Data[23]: 0.000650 0.999180 0.000000 0.000170
Data[24]: 0.000517 0.999348 0.000000 0.000135
Data[25]: 0.013290 0.984035 0.000001 0.002674
Data[26]: 0.000542 0.999317 0.000000 0.000141
Data[27]: 0.000562 0.999292 0.000000 0.000147
Data[28]: 0.000543 0.999315 0.000000 0.000142
Data[29]: 0.000522 0.999342 0.000000 0.000136
Data[30]: 0.000530 0.999332 0.000000 0.000138
Data[31]: 0.011917 0.985108 0.000001 0.002974
Data[32]: 0.000540 0.999319 0.000000 0.000141
Data[33]: 0.995174 0.003988 0.000000 0.000837
Data[34]: 0.000643 0.999189 0.000000 0.000168
Data[35]: 0.000506 0.999362 0.000000 0.000132
Plain Text    Tab Width: 8    Ln 1, Col 1    INS
```

Figure 3.0: Membership matrix

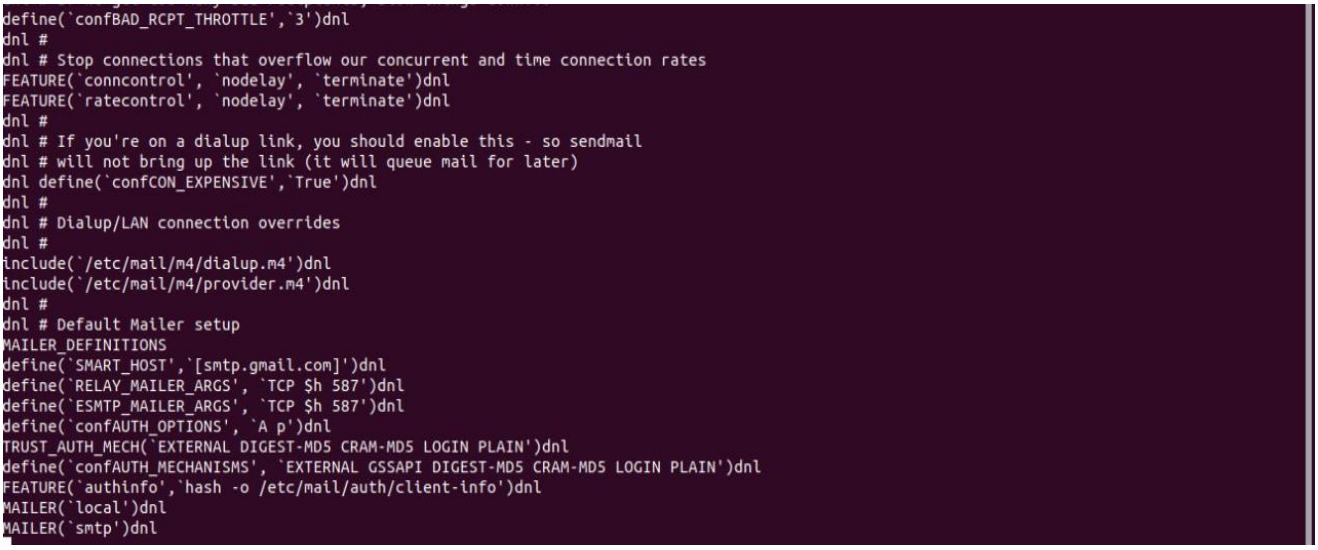
## 3. NETWORK PACKET MONITORING

The network packet monitoring tool used is the bro network packet monitor. It is configured to monitor the network interface for the virtualization manager host. Figure 3.1 shows the network interface and its IP range i.e. 192.168.122.1/24.

```
virbr0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.122.1  netmask 255.255.255.0  broadcast 192.168.122.255
        ether 52:54:00:43:41:53  txqueuelen 1000  (Ethernet)
        RX packets 154250  bytes 10123177 (10.1 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 218681  bytes 364270645 (364.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 3.1: Virtual Manager network interface

Figure 3.2 is the configuration used in initializing the sendmail service and the connection between the sendmail service and the IDS to trigger an email being sent after detection.



```
define(`confBAD_RCPT_THROTTLE',`3')dnl
dnl #
dnl # Stop connections that overflow our concurrent and time connection rates
FEATURE(`conncontrol', `nodelay', `terminate')dnl
FEATURE(`ratecontrol', `nodelay', `terminate')dnl
dnl #
dnl # If you're on a dialup link, you should enable this - so sendmail
dnl # will not bring up the link (it will queue mail for later)
dnl define(`confCON_EXPENSIVE',`True')dnl
dnl #
dnl # Dialup/LAN connection overrides
dnl #
include(`/etc/mail/m4/dialup.m4')dnl
include(`/etc/mail/m4/provider.m4')dnl
dnl #
dnl # Default Mailer setup
MAILER_DEFINITIONS
define(`SMART_HOST',`[smtp.gmail.com]')dnl
define(`RELAY_MAILER_ARGS', `TCP $h 587')dnl
define(`ESMTP_MAILER_ARGS', `TCP $h 587')dnl
define(`confAUTH_OPTIONS', `A p')dnl
TRUST_AUTH_MECH(`EXTERNAL DIGEST-MD5 CRAM-MD5 LOGIN PLAIN')dnl
define(`confAUTH_MECHANISMS', `EXTERNAL GSSAPI DIGEST-MD5 CRAM-MD5 LOGIN PLAIN')dnl
FEATURE(`authinfo',`hash -o /etc/mail/auth/client-info')dnl
MAILER(`local')dnl
MAILER(`smtp')dnl
```

Figure 3.2: Sendmail service Configuration

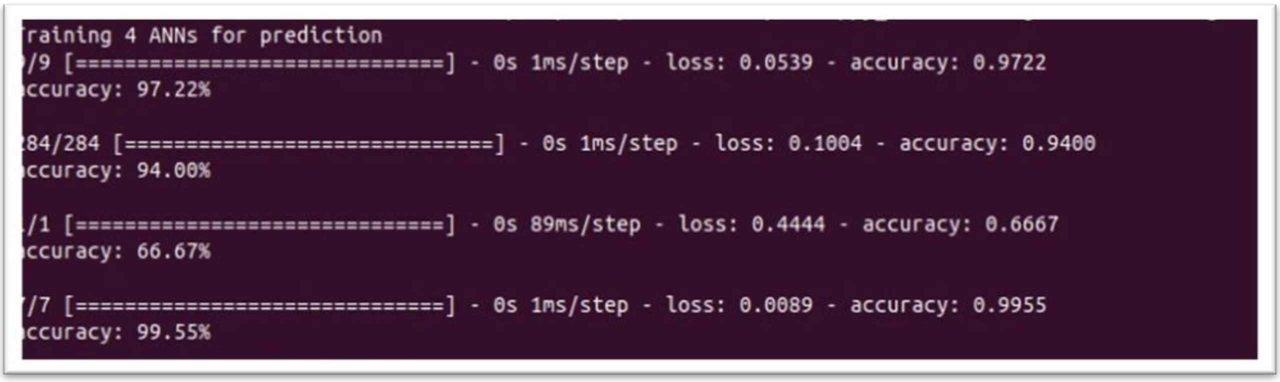
```
[zeek]
type=standalone
host=localhost
interface=virbr0
```

Figure 3.3: Sendmail initialization and the IDS

***3.1. ANN Models***

The architecture of the ANN model comprises of three primary layers, the Input layer, the Hidden layer, and the Output layer. During the input layer phase, the clustered data is extracted from the AFCM output. This data is then fed into the ANN model in the form of a matrix with four layers. To establish the connection between the data and the central point, a function called "calcMembershipValues" is employed. This function calculates the membership value of the input matrix, it explores the intricate relationship between the data and the central point.

The Hidden layer serves as the core engine driving the Intrusion Detection System (IDS) architecture. This layer receives its input from the outcome of the center data file. Leveraging the Keras and TensorFlow libraries, the data is classified into four distinct clusters, signifying different patterns within the data. This pivotal step in the process contributes to the system's ability to discern intricate anomalies and non-conformities in network behavior.



```
raining 4 ANNs for prediction
/9 [==============================] - 0s 1ms/step - loss: 0.0539 - accuracy: 0.9722
ccuracy: 97.22%

84/284 [==============================] - 0s 1ms/step - loss: 0.1004 - accuracy: 0.9400
ccuracy: 94.00%

/1 [==============================] - 0s 89ms/step - loss: 0.4444 - accuracy: 0.6667
ccuracy: 66.67%

/7 [==============================] - 0s 1ms/step - loss: 0.0089 - accuracy: 0.9955
ccuracy: 99.55%
```

Figure 3.4: The four tested model Accuracies

Bipolar Sigmoid is used in the activation function for the output layer, it create an output between the range of 0 and 1. The output is stored in the output directory.

The Virtual environment serves as the cloud environment for deployment of the IDS engine. They are connected and monitored through the Virbr0 interface from the host OS. The IP subnet for the interface is 192.168.122.0/24.

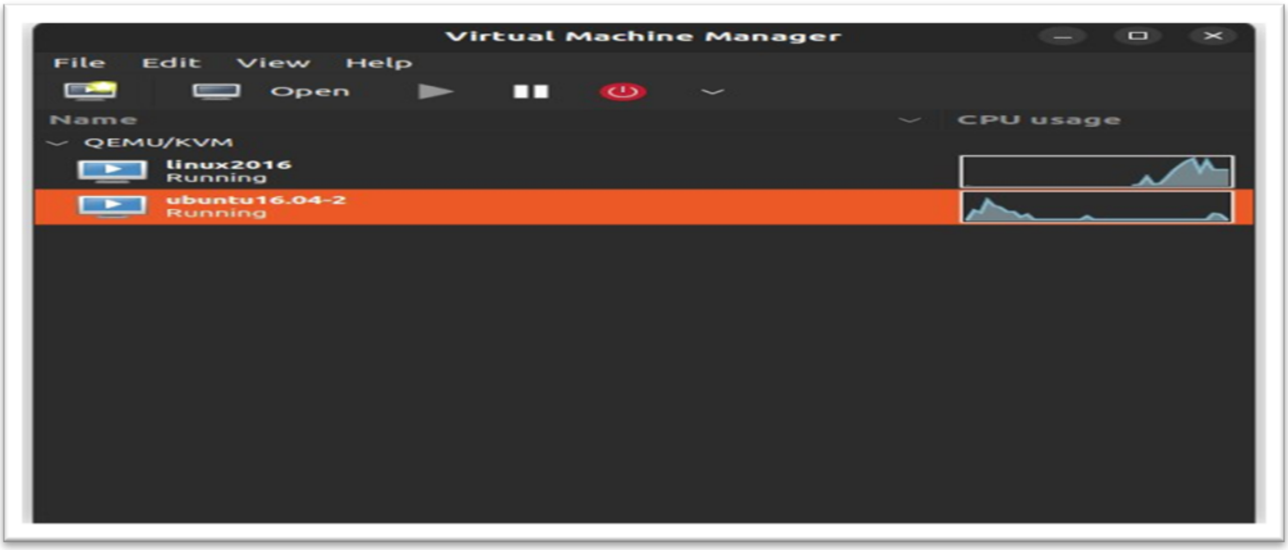



Figure 3.5: QEMU-KVM Libvirt Environment

Figure 3.6 and 3.7 shows the interfaces of the Virtual machines and their corresponding IP address that will be used for testing of the IDS engine. Figure 3.9 is a lightweight Linux distro while Ubuntu 16.04 is a heavyweight Linux distro.

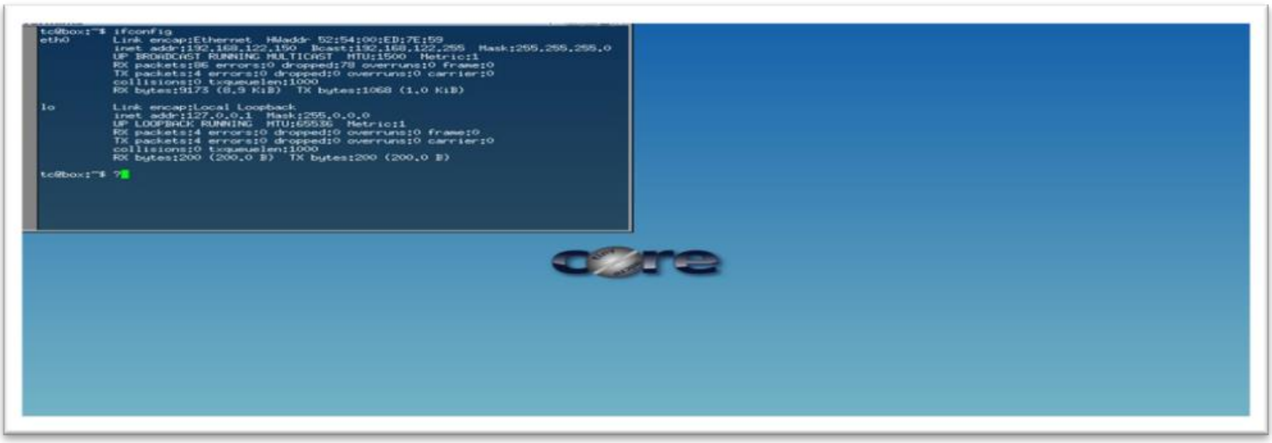


Figure 3.6: Interface of the Tiny core OS for testing

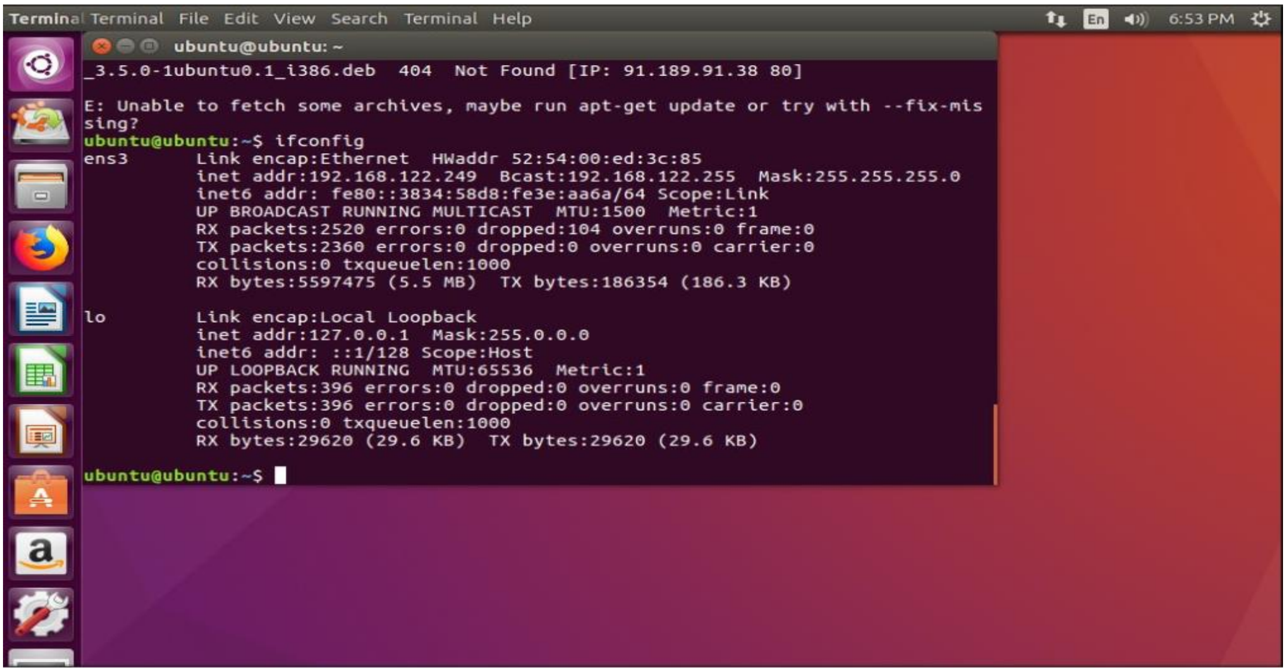



Figure 3.7: Interface on the Ubuntu 16.04 OS

Various tests were carried out to check the functional feature of the IDS within the VM. It is was attacked from outside of the virtual manager i.e. VM trying to attack another VM and normal activity was also tested for. In testing of the mail and log management feature, it was configured to relay mail alerts.

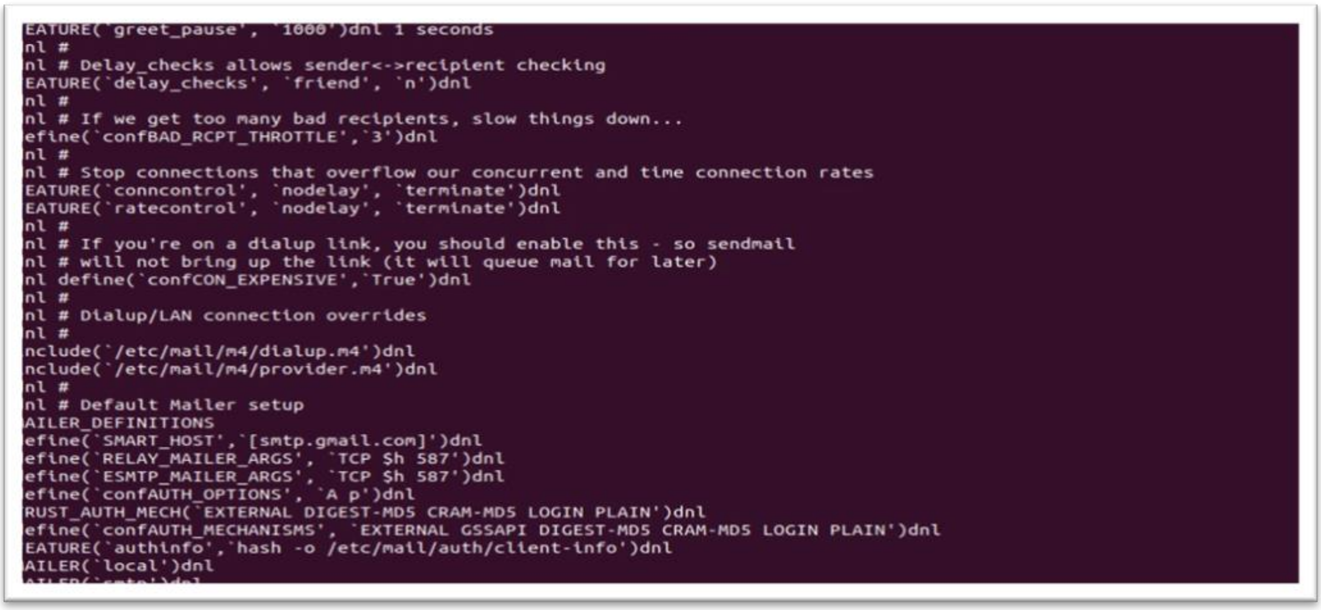



Figure 3.8: GMAIL Relay configurations

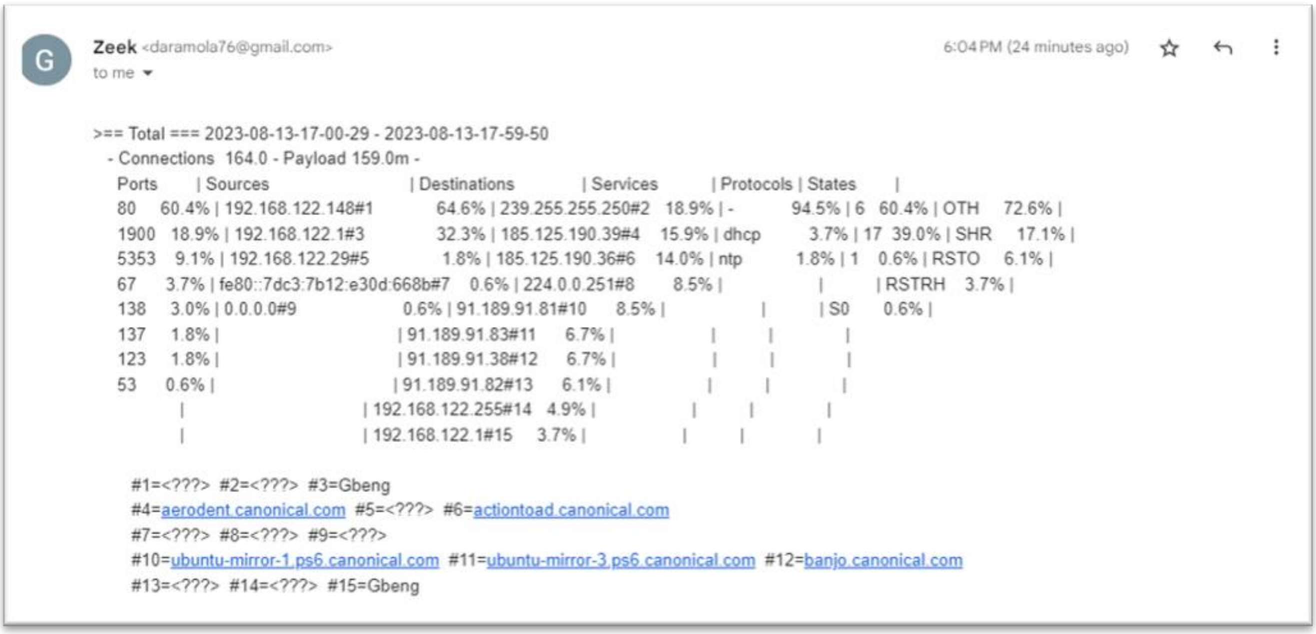



Figure 3.9: IDS detection report email.

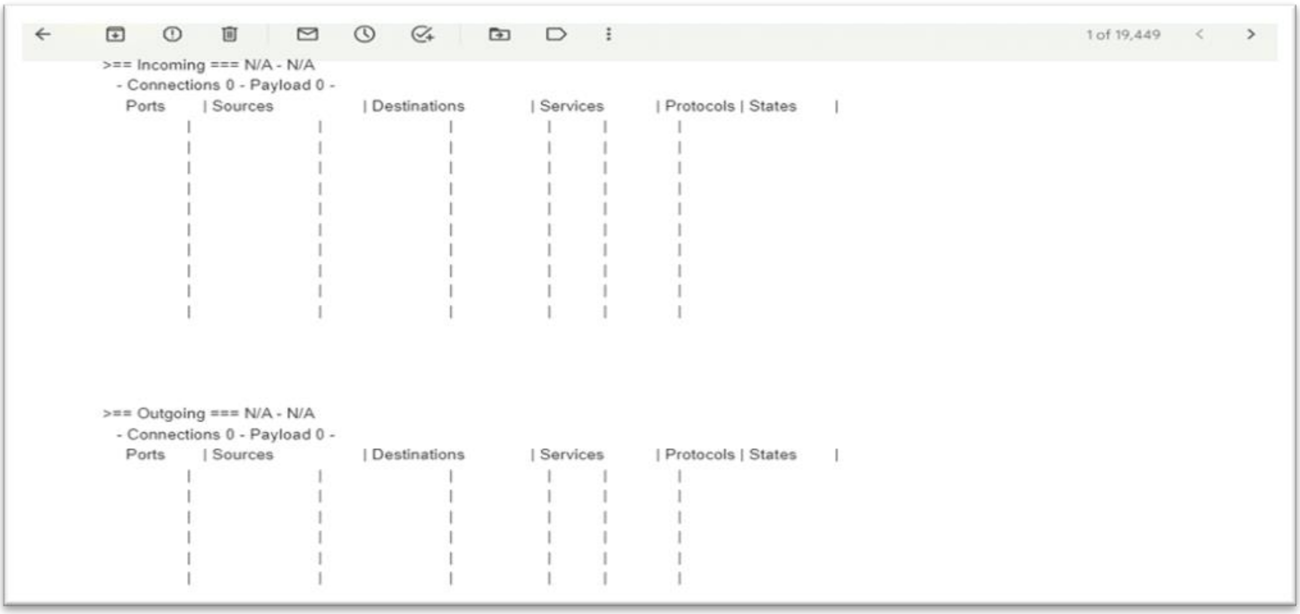



Figure 3.10: Simulation of a normal attack report

## 4. ANALYSIS OF RESULT

The methodical application of these steps creates a strong framework for the creation of a robust anomaly detection system. We are able to fully utilize the capabilities of the AFCM algorithm thanks to our thorough approach to data preparation and cleaning. This project is a crucial part of our comprehensive intrusion detection strategy. Our strategy presents a forward-looking stance in strengthening network security when combined with cutting-edge methodologies like Artificial Neural Networks (ANN). By addressing data complexities and utilizing cutting-edge techniques, a pathway for a proactive approach to safeguard network integrity and resilience against evolving threats is provided.

It uses QEMU Libvirt as the host and artificial neural network (ANN) models to detect intrusions in guest virtual machines (VMs). Developing an accurate intrusion detection system to protect cloud environments was the primary aim of this work. VMs served as guests and QEMU Libvirt served as the host in this two-tiered architecture. Analyzing guest VM activity and spotting potential intrusions was the objective of the detection engine, which was powered by ANN models. The KDD dataset served as the ANN models' training dataset after being thoroughly cleaned. The dataset was improved to include the categories of

Normal, U2R attack, R2L attack, and probe attack, covering a variety of potential threats.

## 5. CONCLUSION

This research successfully developed a robust anomaly detection system for cloud environments by leveraging the Adaptive Fuzzy C-Means (AFCM) algorithm and Artificial Neural Networks (ANN). The meticulous data preparation and cleaning steps facilitated the full utilization of the AFCM algorithm's capabilities, creating a strong framework for detecting anomalies. Our approach is a critical component of a comprehensive intrusion detection strategy, merging cutting-edge methodologies with proactive security measures.

The use of QEMU Libvirt as the host and ANN models for intrusion detection in guest virtual machines (VMs) demonstrated the feasibility and efficiency of our proposed system. By training the ANN models on a cleaned and enhanced KDD dataset, we ensured the system's ability to detect a wide range of potential threats, including Normal, U2R attack, R2L attack, and probe attack categories.

Cloud computing systems are inherently vulnerable to cyber-attacks due to misconfigured services and poor cybersecurity management. To address this, our Multi-Level Intrusion Detection System (IDS) with Log Management offers a balanced solution, maintaining both system resource efficiency and robust security. The integration of the AFCM clustering algorithm with the ANN model training proved effective in enhancing the system's intrusion detection capabilities.

This work highlights the importance of a proactive approach to network security, providing a pathway for safeguarding cloud environments against evolving threats. The proposed architecture not only meets the primary aim of developing an accurate intrusion detection system but also addresses the trade-off between security levels and system performance. By employing a machine learning approach with the KDD CUP'99 dataset, we demonstrated the efficiency and effectiveness of our system compared to existing algorithms.

In conclusion, the successful implementation of our proposed architecture offers a significant advancement in cloud computing security, ensuring the integrity and resilience of network systems. This study paves the way for future research in enhancing intrusion detection systems, emphasizing the importance of innovative methodologies in combating cyber threats.

**Declaration of competing interest:**
Authors declared that there is no conflict of interest regarding this publication.

**Author’s contribution:** All authors contributed equally

## REFERENCES

[1] M. Aminzade, "Confidentiality, integrity and availability – finding a balanced IT framework," Network Security, vol. 2018, no. 5, pp. 9–11, 2018. https://doi.org/10.1016/S1353-4858(18)30043-6

[2] J. Arshad, M. A. Azad, R. Amad, K. Salah, M. Alazab, and R. Iqbal, "A Review of Performance, Energy and Privacy of Intrusion Detection Systems for IoT," Electronics, vol. 9, no. 4, p. 629, 2020. https://doi.org/10.3390/ELECTRONICS9040629

[3] S. Bista and R. Chitrakar, "DDoS Attack Detection Using Heuristics Clustering Algorithm and Naïve Bayes Classification," Journal of Information Security, vol. 9, no. 1, pp. 33–44, 2018. https://doi.org/10.4236/JIS.2018.91004

[4] I. Butun, S. D. Morgera, and R. Sankar, "A survey of intrusion detection systems in wireless sensor networks," IEEE Communications Surveys and Tutorials, vol. 16, no. 1, pp. 266–282, 2014. https://doi.org/10.1109/SURV.2013.050113.00191

[5] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," Computers & Security, vol. 28, no. 1-2, pp. 18–28, 2009. https://doi.org/10.1016/J.COSE.2008.08.003

[6] A. Garg and P. Maheshwari, "A hybrid intrusion detection system: A review," Proceedings of the 10th International Conference on Intelligent Systems and Control, ISCO 2016. https://doi.org/10.1109/ISCO.2016.7726909

[7] R. Gassais, N. Ezzati-Jivan, J. M. Fernandez, D. Aloise, and M. R. Dagenais, "Multi-level host-based intrusion detection system for Internet of things," Journal of Cloud Computing, vol. 9, no. 1, pp. 1–16, 2020. https://doi.org/10.1186/S13677-020-00206-6/TABLES/7

[8] R. Gifty, R. Bharathi, and P. Krishnakumar, "Privacy and security of big data in cyber physical systems using Weibull distribution-based intrusion detection," Neural Computing and Applications, vol. 31, no. 1, pp. 23–34, 2018. https://doi.org/10.1007/S00521-018-3635-6

[9] Y. Han, Y. Du, S. Chen, H. Huang, and Y.-E. Sun, "TSD3: A Novel Time-Series-Based Solution for DDoS Attack Detection," pp. 318–333, 2023. https://doi.org/10.1007/978-3-031-25201-3_25/COVER

[10] I. Idrissi, M. Azizi, and O. Moussaoui, "A Lightweight Optimized Deep Learning-based Host-Intrusion Detection System Deployed on the Edge for IoT," International Journal of Computing and Digital Systems, vol. 11, no. 1, pp. 209–216, 2022. https://doi.org/10.12785/IJCDS/110117

[11] "Intrusion Detection - Rebecca Gurley Bace - Google Books," Retrieved April 2, 2023.

[12] P. Ioulianou, V. Vasilakis, I. Moscholios, and M. Logothetis, "A Signature-based Intrusion Detection System for the Internet of Things," 2018.

[13] N. Kaja, A. Shaout, and D. Ma, "An intelligent intrusion detection system," Applied Intelligence, vol. 49, no. 9, pp. 3235–3247, 2019. https://doi.org/10.1007/S10489-019-01436-1/METRICS

[14] K. Khan, A. Mehmood, S. Khan, M. A. Khan, Z. Iqbal, and W. K. Mashwani, "A survey on intrusion detection and prevention in wireless ad-hoc networks," Journal of Systems Architecture, vol. 105, p. 101701, 2020. https://doi.org/10.1016/J.SYSARC.2019.101701

[15] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," Cybersecurity, vol. 2, no. 1, pp. 1–22, 2019. https://doi.org/10.1186/S42400-019-0038-7/FIGURES/8

[16] A. Khraisat, I. Gondal, P. Vamplew, J. Kamruzzaman, and A. Alazab, "Hybrid Intrusion Detection System Based on the Stacking Ensemble of C5 Decision Tree Classifier and One Class Support Vector Machine," Electronics, vol. 9, no. 1, p. 173, 2020. https://doi.org/10.3390/ELECTRONICS9010173

[17] C. Leghris, O. Elaeraj, and E. Renault, "Improved security intrusion detection using intelligent techniques," Proceedings - 2019 International Conference on Wireless Networks and Mobile Communications, WINCOM 2019. https://doi.org/10.1109/WINCOM47513.2019.8942553

[18] J. Li, Y. Qu, F. Chao, H. P. H. Shum, E. S. L. Ho, and L. Yang, "Machine learning algorithms for network intrusion detection," Intelligent Systems Reference Library, vol. 151, pp. 151–179, 2019. https://doi.org/10.1007/978-3-319-98842-9_6/COVER

[19] C. Liu, Z. Gu, and J. Wang, "A Hybrid Intrusion Detection System Based on Scalable K-Means+ Random Forest and Deep Learning," IEEE Access, vol. 9, pp. 75729–75740, 2021. https://doi.org/10.1109/ACCESS.2021.3082147

[20] N. Moustafa, J. Hu, and J. Slay, "A holistic review of Network Anomaly Detection Systems: A comprehensive survey," Journal of Network and Computer Applications, vol. 128, pp. 33–55, 2019. https://doi.org/10.1016/J.JNCA.2018.12.006

[21] K. Narayana Rao, K. Venkata Rao, and P. R. Prasad, "A hybrid Intrusion Detection System based on Sparseautoencoder and Deep Neural Network," Computer Communications, vol. 180, pp. 77–88, 2021. https://doi.org/10.1016/J.COMCOM.2021.08.026

[22] S. N. Nguyen, V. Q. Nguyen, J. Choi, and K. Kim, "Design and implementation of intrusion detection system using convolutional neural network for DoS detection," ACM International Conference Proceeding Series, pp. 34–38, 2018. https://doi.org/10.1145/3184066.3184089

[23] S. Raza, L. Wallgren, and T. Voigt, "SVELTE: Real-time intrusion detection in the Internet of Things," Ad Hoc Networks, vol. 11, no. 8, pp. 2661–2674, 2013. https://doi.org/10.1016/J.ADHOC.2013.04.014

[24] M. Roesch, "Snort-Lightweight Intrusion Detection for Networks."

[25] Y. K. Saheed, M. O. Arowolo, and A. U. Tosho, "An Efficient Hybridization of K-Means and Genetic Algorithm Based on Support Vector Machine for Cyber Intrusion Detection System," International Journal on Electrical Engineering and Informatics, vol. 14, no. 2, 2022. https://doi.org/10.15676/ijeei.2022.14.2.11

[26] B. M. Serinelli, A. Collen, and N. A. Nijdam, "Training Guidance with KDD Cup 1999 and NSL-KDD Data Sets of ANIDINR: Anomaly-Based Network Intrusion Detection System," Procedia Computer Science, vol. 175, pp. 560–565, 2020. https://doi.org/10.