Configuration Manual

MSc Research Project
MSc Cybersecurity

Navya Tumparthy
Student ID: 23101521

School of Computing
National College of Ireland

Supervisor: Khadija Hafeez

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……Navya Tumparthy……………………………………………………………………………

Student ID: ……23101521…………………………………………………………………………………………

Programme: ……MSc Cybersecurity……………………………… **Year:** 2023-2024……

Module: … MSc Research Practicum part 2……………………………………………………

Lecturer: …… Khadija Hafeez…………………………………………………………………………

Submission Due Date: …12^{th} August 2024………………………………………………………………………

Project Title: … Efficient Intrusion Detection for Smart Homes: Suricata and Machine Learning for Speed and Efficiency…………………………………

Word Count: …2521……………………… **Page Count:**…25……………………………………

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: … Navya Tumparthy……………………………………………………………

Date: ……12^{th} August 2024……………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Navya Tumparthy
23101521

This guide provides the detail steps of integrating a hybrid Machine Learning(ML) model with Suricata an open-source IDS. The steps provided below helps with the reproduction of experimentation as the details of the libraries installed and the scripts used for optimising IDS in smart home application are given.

1 Dataset Cleansing

Step 1: Download the dataset CICIoT2023 from https://www.unb.ca/cic/datasets/iotdataset-2023.html. The dataset contains many CSV files. Each has the network traffic features of different IoT devices. It comprises of both benign and malicious traffic labelled (Neto et al., 2023).
Step 2: Install Anaconda on Mac OS as explained in the link below:
https://docs.anaconda.com/anaconda/install/mac-os/#
Step 3: Identifying the missing and NA values from dataset.

- Open Jupyter notebook from anaconda and create a new notebook for data processing.
- Install pandas library for manipulating dataset.

pip install pandas

- Then run the below script which checks all CSV files for any missing/NA values and provides the counts for each and every class.

```
In [1]: #Libraries that should be imported for working with paths and dataframes
import os
import pandas as pd
from glob import glob

# Path on macOS where the dataset is located
folder_path = '/Users/srinivasm/Downloads/CICIoT'

# As there are multiple CSV files to work glob i used to get all CSV files Get
csv_files = glob(os.path.join(folder_path, '*.csv'))

# As the data is large to ensure optimal working we used chunks of data at a time
def check_na_in_chunk(chunk):
    na_counts = chunk.isna().sum()
    missing_counts = (chunk == '').sum()
    return na_counts, missing_counts

# Counts the values of missing and NA counters
total_na_counts = pd.Series(dtype=int)
total_missing_counts = pd.Series(dtype=int)

# Defining the chunk size
chunk_size = 100000

#Loop for each chunk to identify missing/NA
for file in csv_files:
    try:
        for chunk in pd.read_csv(file, chunksize=chunk_size):
            na_counts, missing_counts = check_na_in_chunk(chunk)
            total_na_counts = total_na_counts.add(na_counts, fill_value=0)
            total_missing_counts = total_missing_counts.add(missing_counts, fill_value=0)
    except Exception as e:
        print(f"Error processing file {file}: {e}")

# Printing the total counts of NA and missing values
print("Total NA counts in the dataset:")
print(total_na_counts)
print("\nTotal missing value counts in the dataset:")
print(total_missing_counts)
```

Step 4: The output of the script shows that there are no missing or NA values in dataset as shown below. If there are any, we need to delete the entries for quality of data.

```
Total NA counts in the dataset:
flow_duration      0.0
Header_Length      0.0
Protocol Type      0.0
Duration           0.0
Rate               0.0
Srate              0.0
Drate              0.0
fin_flag_number    0.0
syn_flag_number    0.0
rst_flag_number    0.0
psh_flag_number    0.0
ack_flag_number    0.0
ece_flag_number    0.0
cwr_flag_number    0.0
ack_count          0.0
syn_count          0.0
fin_count          0.0
urg_count          0.0
rst_count          0.0
HTTP               0.0
HTTPS              0.0
DNS                0.0
Telnet             0.0
SMTP               0.0
SSH                0.0
IRC                0.0
TCP                0.0
UDP                0.0
DHCP               0.0
ARP                0.0
ICMP               0.0
IPv                0.0
LLC                0.0
Tot sum            0.0
Min                0.0
Max                0.0
AVG                0.0
Std                0.0
Tot size           0.0
IAT                0.0
Number             0.0
Magnitue           0.0
Radius             0.0
Covariance         0.0
Variance           0.0
Weight             0.0
label              0.0
dtype: float64

Total missing value counts in the dataset:
flow_duration      0.0
Header_Length      0.0
Protocol Type      0.0
Duration           0.0
Rate               0.0
Srate              0.0
Drate              0.0
fin_flag_number    0.0
syn_flag_number    0.0
rst_flag_number    0.0
psh_flag_number    0.0
ack_flag_number    0.0
ece_flag_number    0.0
cwr_flag_number    0.0
ack_count          0.0
syn_count          0.0
fin_count          0.0
urg_count          0.0
rst_count          0.0
HTTP               0.0
HTTPS              0.0
DNS                0.0
Telnet             0.0
SMTP               0.0
SSH                0.0
IRC                0.0
TCP                0.0
UDP                0.0
DHCP               0.0
ARP                0.0
ICMP               0.0
IPv                0.0
LLC                0.0
Tot sum            0.0
Min                0.0
Max                0.0
AVG                0.0
Std                0.0
Tot size           0.0
IAT                0.0
Number             0.0
Magnitue           0.0
Radius             0.0
Covariance         0.0
Variance           0.0
Weight             0.0
label              0.0
dtype: float64
```

2 Data Balancing

Step 1: The dataset contains multiple classes of attacks. To simplify the dataset first the attacks are classified into 7 types with the help of python scripts. Prior to running the script install the joblib and imbalanced-learn libraries in jupyter notebook (nikitastsinnas, 2024).

```
!pip install joblib
!pip install imbalanced-learn
```

Step 2: Run the script below in jupyter notebook to map all the attack classes into 8 classes (nikitastsinnas, 2024).

```
In [6]: import os
        import pandas as pd
        from glob import glob
        import joblib
        from imblearn.under_sampling import RandomUnderSampler
        from imblearn.over_sampling import SMOTE
        from imblearn.pipeline import Pipeline

        # Define the path to your folder containing the CSV files
        folder_path = '/Users/srinivasm/Downloads/CICIoT'
        new_dataset_path = '/Users/srinivasm/Downloads/New_CICIoT.csv'

        # Get all CSV files in the folder
        csv_files = glob(os.path.join(folder_path, '*.csv'))

        def category_extraction(df):
            # Extract attack category from label
            category_dict = {
                'DDoS-ACK_Fragmentation' : 'DDoS',
                'DDoS-HTTP_Flood' : 'DDoS',
                'DDoS-ICMP_Flood': 'DDoS',
                'DDoS-PSHACK_Flood': 'DDoS',
                'DDoS-RSTFINFlood': 'DDoS',
                'DDoS-SYN_Flood': 'DDoS',
                'DDoS-SlowLoris': 'DDoS',
                'DDoS-SynonymousIP_Flood': 'DDoS',
                'DDoS-TCP_Flood': 'DDoS',
                'DDoS-UDP_Flood': 'DDoS',
                'DDoS-UDP_Fragmentation': 'DDoS',
                'DDoS-ICMP_Fragmentation' : 'DDoS',

                'DoS-HTTP_Flood' : 'DoS',
                'DoS-SYN_Flood' : 'DoS',
                'DoS-TCP_Flood' : 'DoS',
                'DoS-UDP_Flood' : 'DoS',
                'DictionaryBruteForce' : 'BruteForce',

                'MITM-ArpSpoofing' : 'Spoofing',
                'DNS_Spoofing' : 'Spoofing',

                'Recon-HostDiscovery' : 'Recon',
                'Recon-OSScan' : 'Recon',
                'Recon-PingSweep' : 'Recon',
                'Recon-PortScan' : 'Recon',
                'VulnerabilityScan' : 'Recon',

                'SqlInjection' : 'Web-based',
                'CommandInjection' : 'Web-based',
                'Backdoor_Malware' : 'Web-based',
                'Uploading_Attack' : 'Web-based',
                'XSS' : 'Web-based',
                'BrowserHijacking' : 'Web-based',

                'Mirai-greeth_flood' : 'Mirai',
                'Mirai-greip_flood' : 'Mirai',
                'Mirai-udpplain' : 'Mirai',

                'BenignTraffic' : 'Benign'
            }

            # Label encoding for attack categories
            df_label_cat = df.label.apply(lambda x: category_dict.get(x))
            df['label'] = df_label_cat
            return df
```

- Create a function to analyse the data imbalance and then balance it using under sampling and SMOT methods as in below screenshot (nikitastsinnas, 2024).

```python
def csvToBalancedDataset(first, last, folder_path):
    balanced_dfs = []
    csv_files = glob(os.path.join(folder_path, '*.csv'))
    for index, file in enumerate(csv_files[first:last], start=first):
        try:
            df = pd.read_csv(file)
            # Change column names
            df.columns = ['_'.join(c.split(' ')).lower() for c in df.columns]

            # Drop NULLs & reset index
            df.dropna(inplace=True)
            df.reset_index(inplace=True, drop=True)

            # Extract binary labels
            df = category_extraction(df)

            # Balance the classes in each dataframe
            balanced_dfs.append(df)
        except Exception as e:
            print(f"Error processing file {file}: {e}")

    return pd.concat(balanced_dfs, axis=0, ignore_index=True)

# Load and preprocess the data
df = csvToBalancedDataset(0, 50, folder_path)
```

- Run commands below to check the data balance and the output shows quite an imbalance.

```python
df = csvToBalancedDataset(0,50)
df['label'].value_counts()
```

- The output should shows like below where there is great imbalance between classes (nikitastsinnas, 2024).

```
label
DDoS          9361472
DoS           2227900
Mirai          725551
Benign         302896
Spoofing       134176
Recon           97110
Web-based        6850
BruteForce       3590
Name: count, dtype: int64
```

- To address the imbalance, under sample the higher count data and oversample the lower count data to 10,000 counts by using the below script and save the new dataset as ‘New_CICIoT.csv’ (nikitastsinnas, 2024).

```python
# Load and preprocess the data
df = csvToBalancedDataset(0, 50, folder_path)

# Separate features and labels
X = df.drop(columns=['label'])
y = df['label']

#  undersampling and oversampling
under = RandomUnderSampler(sampling_strategy={'Benign': 10000, 'DDoS': 10000, 'DoS': 10000, 'Mirai': 10000, 'Spoofin
over = SMOTE(sampling_strategy={'Web-based': 10000, 'BruteForce': 10000})

# Combine undersampling and oversampling in a pipeline
pipeline = Pipeline(steps=[('under', under), ('over', over)])

# pipeline to balance the dataset
X_resampled, y_resampled = pipeline.fit_resample(X, y)

# Combine resampled features and labels into a new DataFrame
balanced_df = pd.concat([pd.DataFrame(X_resampled), pd.DataFrame(y_resampled, columns=['label'])], axis=1)

# Save the balanced dataset to a new CSV file
balanced_df.to_csv(new_dataset_path, index=False)

# Check the balance of the new dataset
print("Balanced data:")
print(balanced_df['label'].value_counts())
```

```
Balanced data:
Spoofing      10000
Recon         10000
BruteForce    10000
DDoS          10000
DoS           10000
Benign        10000
Mirai         10000
Web-based     10000
Name: label, dtype: int64
```

Note: This script can be used together in a single jupyter notebook cell.

3 Model Training and Evaluation

As the data is pre-processed and transformed into new dataset 'New_CICIoT.csv' this can be used for model training.

Step 1: Open Google Colab a free computing service from google to train and evaluate models with ease rather using our own computing resources (research.google.com, 2023). Upload the transformed dataset to the colab.

Step 2: Install below packages before model training

```
pip install pandas
pip install scikit-learn
pip install lightgbm
pip install joblib
pip install seaborn
pip install matplotlib
```

Step 3: Run the entire script together to train and save the model.

- First, the necessary libraries are imported for model training and the balanced dataset is loaded.
- Later the dataset is split into training and testing in proportion of 70-30 and then RF model is initialised with appropriate hyper-parameter tuning by analysing the accuracy as below.

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
import lightgbm as lgb
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, roc_curve, auc
import joblib
import seaborn as sns
import matplotlib.pyplot as plt
import time

# Load the balanced dataset
new_dataset_path = 'New_CICIoT.csv'
df = pd.read_csv(new_dataset_path)

# Separate features and labels
X = df.drop(columns=['label'])
y = df['label']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Train a RF model
rf = RandomForestClassifier(
    n_estimators=100,
    max_depth=10,
    min_samples_split=10,
    min_samples_leaf=10,
    bootstrap=True,
    random_state=42,
    class_weight='balanced'
```

- Then the top ranked features as per RF built-in functionality are extracted and printed out for next layer ML training.
- LGBM is trained by adding hyper-parameters and cross-validation of 5 splits.
- Later the model is tested using classification report for training and testing data.

```python
)

rf.fit(X_train, y_train)

# RF feature importance
feature_importances = pd.Series(rf.feature_importances_, index=X.columns).sort_values(ascending=False)
print(feature_importances)

# Select top features based on importance
top_features = feature_importances.head(35).index

# Defining LightGBM with hyper parameter tuning
lgbm = lgb.LGBMClassifier(
    num_leaves=20,  # Reduce the number of leaves
    max_depth=5,  # Reduce the maximum depth
    learning_rate=0.05,
    n_estimators=1000,  # Increase number of trees
    min_child_samples=100,
    reg_alpha=10.0,  # Further increase L1 regularization
    reg_lambda=10.0,  # Further increase L2 regularization
    feature_fraction=0.8,  # Use feature subsampling
    bagging_fraction=0.8,  # Use data subsampling
    bagging_freq=1,  # Perform bagging at every iteration
    random_state=42,
    class_weight='balanced'
)

# Implement early stopping using callbacks
early_stopping_callback = lgb.early_stopping(stopping_rounds=50, verbose=True)

#  cross-validation
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

cv_scores = cross_val_score(lgbm, X_train[top_features], y_train, cv=skf, scoring='accuracy')
print(f"Cross-validation scores: {cv_scores}")
print(f"Mean cross-validation score: {cv_scores.mean()}")

# Train LightGBM with top 20 features selected by RF
lgbm.fit(
    X_train[top_features], y_train,
    eval_set=[(X_test[top_features], y_test)],
    eval_metric='logloss',
    callbacks=[early_stopping_callback]
)

# Validate the model on training data
y_train_pred = lgbm.predict(X_train[top_features])
y_test_pred = lgbm.predict(X_test[top_features])

# Evaluating model performance on training data
print("Training Set Classification Report:")
print(classification_report(y_train, y_train_pred))
train_accuracy = accuracy_score(y_train, y_train_pred)
print(f"Training Accuracy: {train_accuracy:.4f}")

# Evaluating model performance on test data
print("Testing Set Classification Report:")
print(classification_report(y_test, y_test_pred))
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f"Testing Accuracy: {test_accuracy:.4f}")
```

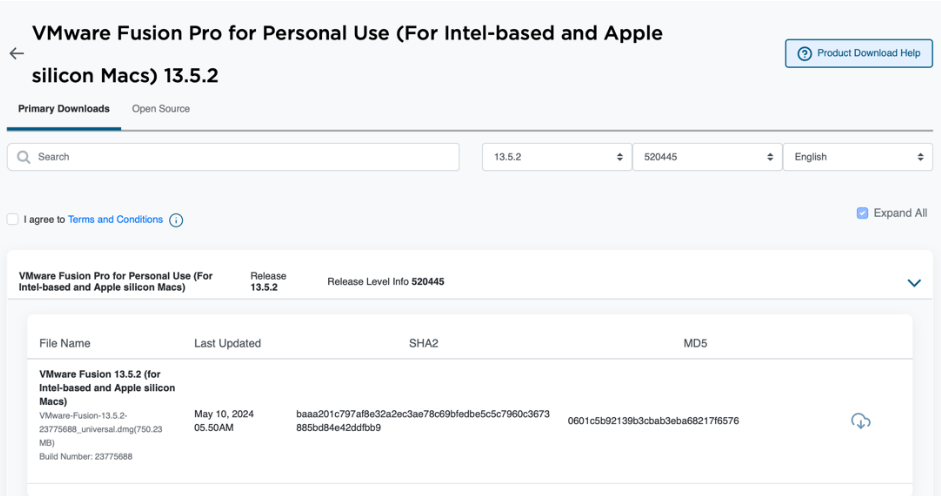
- Difference between the accuracy of training and testing data is calculated to check overfitting problem.
- Later Confusion matrix and ROC AUC Curves are printed for analysing the model's performance in visual manner.
- The model is saved into pkl file which is downloaded from google colab for implementation purpose.

```python
# Checking overfitting
if abs(train_accuracy - test_accuracy) > 0.05:
    print("Warning: The model may be overfitting.")
else:
    print("The model does not appear to be overfitting.")

# Confusion Matrix creation
conf_matrix = confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=y.unique(), yticklabels=y.unique())
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# ROC and AUC for each class
y_test_bin = pd.get_dummies(y_test)
y_test_pred_bin = pd.get_dummies(y_test_pred)

plt.figure(figsize=(10, 7))
for i in range(len(y.unique())):
    fpr, tpr, _ = roc_curve(y_test_bin.iloc[:, i], y_test_pred_bin.iloc[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=2, label=f'ROC curve (area = {roc_auc:.2f}) for class {y.unique()[i]}')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Saving the trained model
model_filename = 'lgbm_model_top_features_regularized.pkl'
joblib.dump(lgbm, model_filename)

print(f"Model saved to {model_filename}")
```

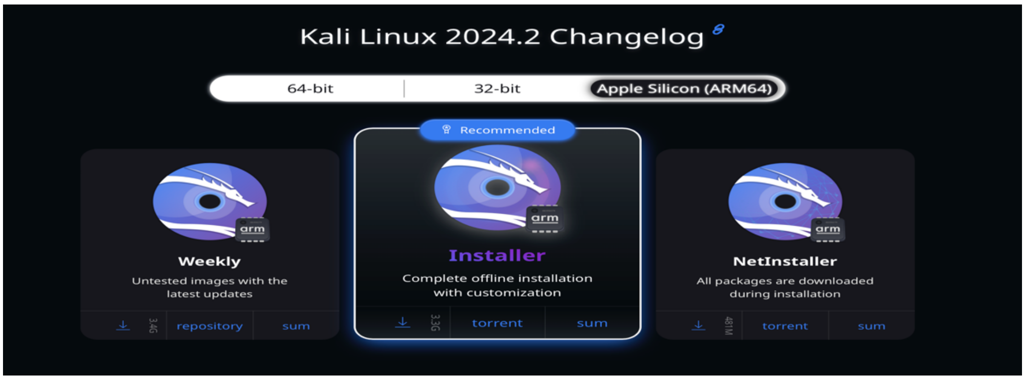
4 Lab Setup

After the training and evaluating model for performance, next goal is to implement the model in simulated smart home by integrating with Suricata an open-source signature-based IDS and check its efficiency in terms of identifying the attack, computational power used for detection and speed of detection. To achieve this there should be a proper lab setup.

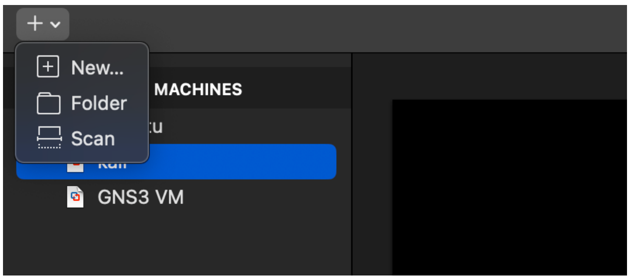
- Install VMware fusion pro version 13.5.2 on your MAC Book Pro from link below. One need to register for downloading. But it is a free service for personal use. https://support.broadcom.com/group/ecx/productdownloads?subfamily=VMware+Fusion



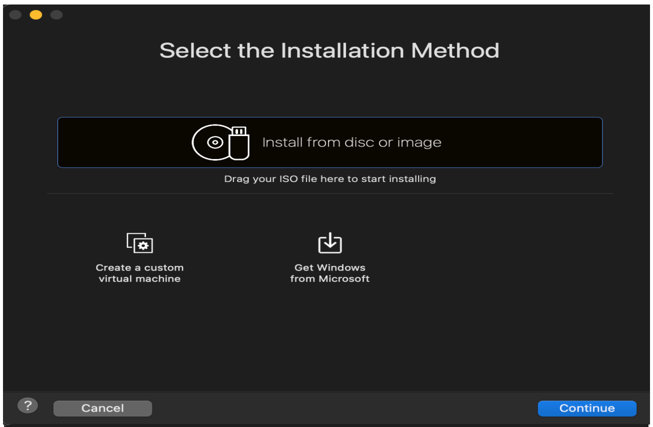
- After installation download Kali Linux from https://www.kali.org/get-kali/#kali-installer-images
- Select as Apple Silicon as shown below, and an ISO file of Kali Linux will be downloaded.

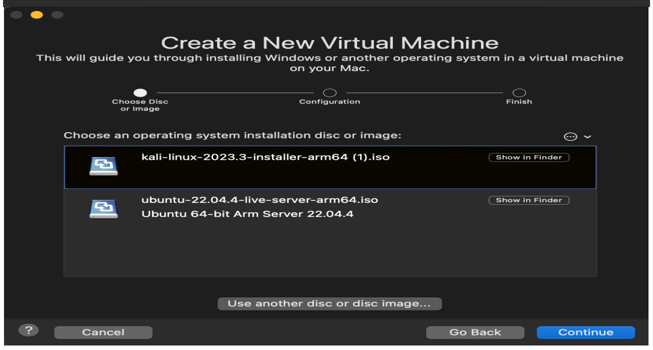


- Open VMware application and click on '+' symbol to add a new Virtual Machine to your environment.

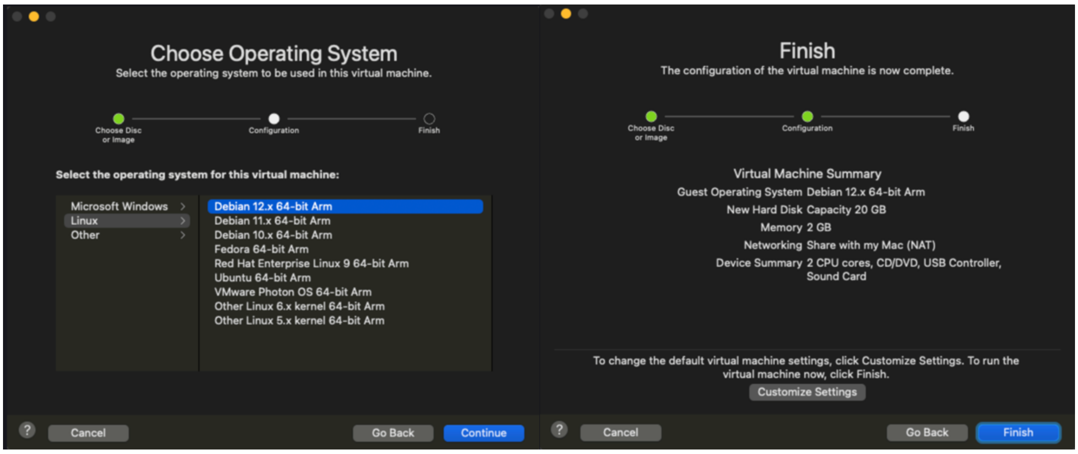


- Then select the 'Install from disc or image' option and then select downloaded Kali Linux ISO image then proceed further.

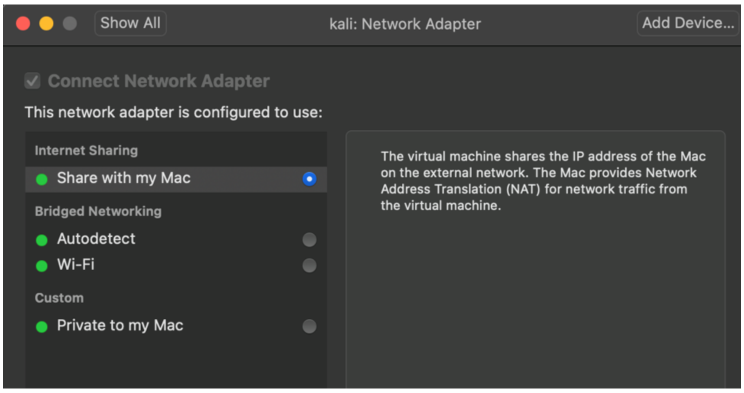




- Select Debian's latest version as Operating System and then provide 2GB of RAM, 2 Core processors and 20 GB ROM during installation and proceed installing the Kali Linux.



- Create a username and password for the Kali VM for security.
- Ubuntu server of version 22.04.4 LTS (Jammy Jellyfish) of 64-bit ARM architecture is used for MAC OS, which can be downloaded from https://cdimage.ubuntu.com/releases/22.04/release/
- Repeat the same steps to create one more VM with Ubuntu ISO image by providing 4GB RAM, 2 core processors and 50GB ROM.
- Proceed with installation by setting up username and password.
- Ensure the network adapter setting for both VM's is set to 'NAT' as this enables the communication between the VM's and VM's to internet.



- Ensure to take snapshots of VM's before starting the experimentation.
- By Default, Ubuntu server comes with CLI and there is no GUI. To ease the usage lightweight GUI is installed. LightDM is used for GUI which is explained in https://roman-academy.medium.com/how-to-install-a-desktop-environment-gui-in-ubuntu-server-66b131d4da8c

5 Suricata Installation

- Open and login to Ubuntu Server for installing Suricata. Suricata version 7.0.6 was installed during experimentation (docs.suricata.io, 2024).
- First update your system to ensure latest packages are available and also with all dependencies for smooth working of Suricata (docs.suricata.io, 2024).

```
sudo apt update
sudo apt upgrade -y
Install the libraries are are necessary for Suricata installation
Sudo apt install -y libjansson, libpcap, libpcre2, libyaml, zlib
Sudo apt install -y make gcc  pkg-config rustc cargo
sudo apt-get install autoconf automake build-essential ccache clang
curl git \
        gosu jq libbpf-dev libcap-ng0 libcap-ng-dev libelf-dev \
        libevent-dev libgeoip-dev libhiredis-dev libjansson-dev \
        liblua5.1-dev libmagic-dev libnet1-dev libpcap-dev \
        libpcre2-dev libtool libyaml-0-2 libyaml-dev m4 make \
        pkg-config python3 python3-dev python3-yaml sudo zlib1g \
        zlib1g-dev
cargo install --force cbindgen
sudo apt-get install software-properties-common
sudo add-apt-repository ppa:oisf/suricata-stable
sudo apt-get update
sudo apt-get install suricata
```

- Check if the directories are present by running commands below:

```
sudo cd /etc/suricata/rules
sudo cd /var/lib/suricata
sudo cd /var/log/suricata
```

- If directories are not created create directories by using below commands:

```
sudo mkdir -p /etc/suricata/rules
```

sudo mkdir /var/lib/suricata
sudo mkdir /var/log/suricata

- Once the directories are present update the rules of Suricata with the latest signatures:

sudo suricata-update

- Now enable Suricata and start the service.

sudo systemctl enable suricata
sudo systemctl start suricata

- Check if it is running successfully by the command below and it should show that the service is running (docs.suricata.io, 2024).

sudo systemctl status suricata

6 Mininet Installation

- First upgrade your ubuntu packages

sudo apt update
sudo apt upgrade -y

- Later run the below command to install mininet directly from packages of ubuntu (Mininet Team, 2018).

sudo apt-get install mininet

- To check version of mininet use command below

mn –version

- To manage mininet using controller download pox controller (Mininet Team, 2018)

git clone https://github.com/noxrepo/pox.git
cd pox
./pox.py forwarding.l2_learning

```
navya@navya:~$ cd pox/
navya@navya:~/pox$ ./pox.py forwarding.l2_learning
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
```

7 Smart Home Simulation

- Smart home simulation using mininet python scripting. Below script explains that the five nodes are simulated for smart home and are connected with each other. IPs has been assigned to all the nodes (Mininet Team, 2018).
- Virtual ethernet has been created to have communication between smart home and Kali VM

```python
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSKernelSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import Intf, TCLink

def smartHomeTopo():
    net = Mininet(controller=RemoteController, switch=OVSKernelSwitch, autoSetMacs=True)

    info('*** Adding controller\n')
    net.addController('c0', ip='127.0.0.1', port=6633)

    info('*** Adding smart home devices\n')
    light1 = net.addHost('light1', ip='10.0.0.1/24')
    light2 = net.addHost('light2', ip='10.0.0.2/24')
    thermostat = net.addHost('thermostat', ip='10.0.0.3/24')
    camera = net.addHost('camera', ip='10.0.0.4/24')
    hub = net.addHost('hub', ip='10.0.0.5/24')  # Smart home hub

    info('*** Adding switch\n')
    s1 = net.addSwitch('s1')

    info('*** Creating links\n')
    net.addLink(light1, s1)
    net.addLink(light2, s1)
    net.addLink(thermostat, s1)
    net.addLink(camera, s1)
    net.addLink(hub, s1)

    info('*** Adding virtual Ethernet pair to switch\n')
    Intf('veth1', node=s1)

    info('*** Starting network\n')
    net.start()

    info('*** Installing tools on smart home devices\n')
    devices = [light1, light2, thermostat, camera, hub]
    for device in devices:
        device.cmd('apt-get update')
        device.cmd('apt-get install -y curl iperf dnsutils hping3')
    info('*** Configuring routes on smart home devices\n')
    light1.cmd('ip route add default via 10.0.0.5')
    light2.cmd('ip route add default via 10.0.0.5')
    thermostat.cmd('ip route add default via 10.0.0.5')
    camera.cmd('ip route add default via 10.0.0.5')
    hub.cmd('ip route add default via 10.0.0.5')

    info('*** Adding route on the hub to reach external network\n')
    hub.cmd('ip route add 192.168.30.0/24 via 10.0.0.100')

    info('*** Generating traffic to simulate smart home activity\n')
    # Simulate ICMP traffic
    light1.cmd('hping3 -1 10.0.0.2 -c 5 &')
    thermostat.cmd('hping3 -1 10.0.0.4 -c 5 &')

    # Simulate HTTP/HTTPS traffic
    light2.cmd('curl http://10.0.0.4 &')
    camera.cmd('curl https://10.0.0.5 &')

    # Simulate DNS queries (assuming hub can resolve DNS queries)
    thermostat.cmd('nslookup google.com 10.0.0.5 &')

    # Simulate TCP traffic with iperf
    hub.cmd('iperf -s &')
    light1.cmd('iperf -c 10.0.0.5 -t 10 &')

    # Simulate UDP traffic with iperf
    camera.cmd('iperf -u -c 10.0.0.5 -t 10 &')

    info('*** Running CLI\n')
    CLI(net)

    info('*** Stopping network\n')
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    smartHomeTopo()
```

- To run the simulation run the script using python3 and it should connect the smart home as below.

```
navya@navya:~$ sudo python3 smarthome1.py
*** Adding controller
*** Adding smart home devices
*** Adding switch
*** Creating links
*** Adding virtual Ethernet pair to switch
*** Starting network
*** Configuring hosts
light1 light2 thermostat camera hub
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Configuring routes on smart home devices
*** Adding route on the hub to reach external network
*** Running CLI
*** Starting CLI:
mininet>
```

- To test if all devices are working as expected just run pingall in mininet command line and result should show 0% drop in packets.

```
mininet> pingall
*** Ping: testing ping reachability
light1 -> light2 thermostat camera hub
light2 -> light1 thermostat camera hub
thermostat -> light1 light2 camera hub
camera -> light1 light2 thermostat hub
hub -> light1 light2 thermostat camera
*** Results: 0% dropped (20/20 received)
mininet>
```

- As this smart home should be monitored by Suricata to check the interfaces of the devices simulated run command 'sh ifconfig' as shown below (Mininet Team, 2018).

```
mininet> sh ifconfig
ens160: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.30.11  netmask 255.255.255.0  broadcast 192.168.30.255
        inet6 fe80::20c:29ff:fe38:9d1b  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:38:9d:1b  txqueuelen 1000  (Ethernet)
        RX packets 123  bytes 11973 (11.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 173  bytes 17485 (17.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 46  memory 0x3fe00000-3fe20000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2074  bytes 234118 (234.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2074  bytes 234118 (234.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::b0fd:aaff:fe5b:bf09  prefixlen 64  scopeid 0x20<link>
        ether b2:fd:aa:5b:bf:09  txqueuelen 1000  (Ethernet)
        RX packets 26  bytes 1916 (1.9 KB)
```

```
s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::2c5b:f7ff:fec4:ede3  prefixlen 64  scopeid 0x20<link>
        ether 2e:5b:f7:c4:ed:e3  txqueuelen 1000  (Ethernet)
        RX packets 25  bytes 1846 (1.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 87  bytes 7556 (7.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::1451:2fff:fe00:6e5d  prefixlen 64  scopeid 0x20<link>
        ether 16:51:2f:00:6e:5d  txqueuelen 1000  (Ethernet)
        RX packets 26  bytes 1916 (1.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 86  bytes 7506 (7.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth4: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::fc87:41ff:feea:ca3e  prefixlen 64  scopeid 0x20<link>
        ether fe:87:41:ea:ca:3e  txqueuelen 1000  (Ethernet)
        RX packets 25  bytes 1846 (1.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 86  bytes 7486 (7.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
s1-eth5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::c047:d1ff:fee3:9334  prefixlen 64  scopeid 0x20<link>
        ether c2:47:d1:e3:93:34  txqueuelen 1000  (Ethernet)
        RX packets 25  bytes 1846 (1.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 86  bytes 7486 (7.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

veth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.100  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::3084:71ff:fe2e:138  prefixlen 64  scopeid 0x20<link>
        ether 32:84:71:2e:01:38  txqueuelen 1000  (Ethernet)
        RX packets 76  bytes 6625 (6.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 44  bytes 5416 (5.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

veth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::447e:53ff:fe16:a629  prefixlen 64  scopeid 0x20<link>
        ether 46:7e:53:16:a6:29  txqueuelen 1000  (Ethernet)
        RX packets 44  bytes 5416 (5.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 76  bytes 6625 (6.6 KB)
```

- Based on the above results we could see 5 interfaces and virtual ethernet tunnels. These five interfaces should be added in suricata.yaml file for monitoring.
- To ensure smart home networks connectivity with ubuntu as well as kali VM we have used virtual ethernet and IP forwarding. For which below changes are performed in ubuntu machine.

sudo ip link add veth0 type veth peer name veth1
sudo ip addr add 10.0.0.100/24 dev veth0
sudo ip link set veth0 up
sudo sysctl -w net.ipv4.ip_forward=1
sudo iptables -t nat -A POSTROUTING -o ens160 -j MASQUERADE
sudo iptables -A FORWARD -i ens160 -o veth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
sudo iptables -A FORWARD -i veth0 -o ens160 -j ACCEPT
sudo ip link set veth1 up

- The route should be added in kali VM as well.

sudo ip route add 10.0.0.0/24 via 192.168.30.11

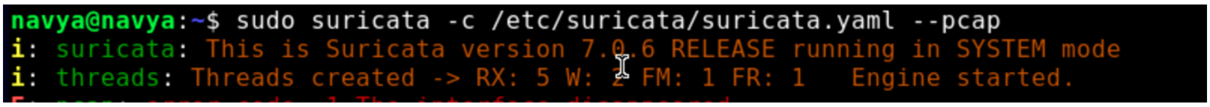
8 Suricata Setup

- To monitor the smart home interfaces open suricata.yaml file located at /etc/suricata and add interfaces under ‘pcap’ section as shown below (docs.suricata.io, 2024):

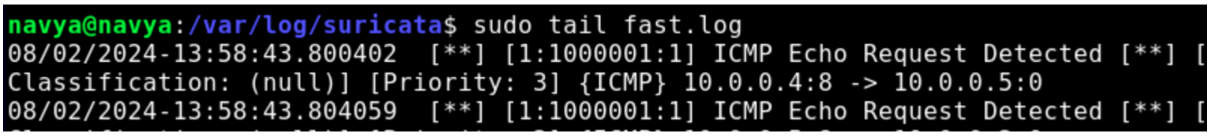
```
navya@navya:/etc/suricata$ ls
classification.config  reference.config  suricata_bkup.yaml  suricata.yaml  threshold.config
navya@navya:/etc/suricata$
```

```
# Cross platform libpcap capture support
pcap:
  #- interface: ens160
  - interface: s1-eth1
  - interface: s1-eth2
  - interface: s1-eth3
  - interface: s1-eth4
  - interface: s1-eth5
    # On Linux, pcap will try to use mmap'ed capture and will use "buffer-size"
    # as total memory used by the ring. So set this to something bigger
    # than 1% of your bandwidth.
    #buffer-size: 16777216
```

- Run pox controller and smart home network first. Later run Suricata using below command and the output show that the ‘Engine is Started’ meaning it is monitoring the interfaces we have added.



- The attacks can be viewed in fast.log located at /var/log/suricata directory as below. To test we added a rule to detect ICMP traffic in rules folder located at /var/lib/suricata/rules and the output shows like below.



- To capture the traffic that is necessary for ML model prediction below traffic rules are added in suricata.yaml file to capture in eve.json logs located at /var/log/suricata (docs.suricata.io, 2024) and it should look like below:

```
- eve-log:
    enabled: yes
    filetype: regular #regular|syslog|unix_dgram|unix_stream|redis
    filename: eve.json
    # include the name of the input pcap file in pcap file processing mode
    pcap-file: false
    community-id: false
    # Seed value for the ID output. Valid values are 0-65535.
    community-id-seed: 0
    xff:
      enabled: no
      mode: extra-data
      deployment: reverse
      header: X-Forwarded-For
    types:
      - alert:
          tagged-packets: yes
          fields: [timestamp, src_ip, dest_ip, src_port, dest_port, proto, flow_id, in_iface,
event_type, alert.severity, alert.signature]
      - frame:
          # disabled by default as this is very verbose.
          enabled: no
      - anomaly:
          enabled: yes
          types:
            # decode: no
            # stream: no
            # applayer: yes
          #packethdr: no
```

```yaml
      - http:
          extended: yes
      - dns:
          enabled: yes
          query: yes
          answer: yes
      - tls:
          extended: yes
      - files:
          force-magic: no
      - smtp:
          extended: yes
      - ftp
      - rdp
      - nfs
      - smb
      - tftp
      - ike
      - dcerpc
      - krb5
      - bittorrent-dht
      - snmp
      - rfb
      - sip
      - dhcp:
          enabled: yes
          extended: no
      - ssh
      - mqtt:
          # passwords: yes           # enable output of passwords
          enabled: yes
      - http2
      - pgsql:
          enabled: no
          # passwords: yes           # enable output of passwords. Disabled by default
      - stats:
          totals: yes       # stats for all threads merged together
          threads: no       # per thread stats
          deltas: no        # include delta values
      - flow:
          fields: [flow_id, timestamp, flow_duration, protocol, src_ip, dest_ip, src_port, dest_port, bytes_toclient, bytes_toserver, packets_toclient, packets_toserver, start, end, age, state]
      - netflow:
```

```
enabled: yes
```

9 ML Integration

- In this next step we integrate ML model with Suricata. This is done by using python scripting where eve.json file is parsed for necessary features.
- Save the script in a file and run it along with Mininet network and Suricata.
- The script is saved as 'pythonmonitoring.py' file.
- To run the script, use the command below and it should run with no errors.

```
python3 pythonmonitoring.py
```

- First all the libraries needed for script are imported.
- Model is loaded using joblib.
- Path for eve.json is written for accessing the logs along with fast.log.
- Interfaces that should be monitored are specified for better understanding.
- To analyse the CPU utilisation process ID of Suricata and python script are provided which should be checked in your systems while running the scripts and change it accordingly.
- To ensure the accuracy of model's performance the mean values of the features are used in case if the feature is empty from eve.json

```
import joblib
import numpy as np
import pandas as pd
import psutil
from datetime import datetime
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler

# Load the trained model
lgbm_model = joblib.load('lgbm_model_top_features_regularized.pkl')

# Paths to the log files
EVE_JSON_PATH = '/var/log/suricata/eve.json'
FAST_LOG_PATH = '/var/log/suricata/fast.log'

# Interfaces to monitor
INTERFACES = ["s1-eth1", "s1-eth2", "s1-eth3", "s1-eth4", "s1-eth5"]

# Track the initial position in eve.json
initial_eve_json_position = 0

# PIDs for resource monitoring
SURICATA_PID = 3303
ML_SCRIPT_PID = psutil.Process().pid  # Get the current script's PID

# Mean values for features
mean_values = {
    "iat": 83182525.9,
    "magnitue": 13.12182,
    "header_length": 76705.9637,
    "rst_count": 38.4681213,
    "protocol_type": 9.06568989,
    "avg": 124.668815,
    "max": 181.963418,
    "tot_size": 124.691567,
    "urg_count": 6.23982356,
    "variance": 0.0964376,
    "tot_sum": 1308.32257,
    "min": 91.6073456,
    "flow_duration": 5.76544939,
    "syn_count": 0.33035785,
    "srate": 9064.05724,
    "rate": 9064.05724,
```

- A function is used to initialise a data structure for extracting all features that are necessary for models' prediction.

```
    "syn_count": 0.33035785,
    "srate": 9064.05724,
    "rate": 9064.05724,
    "radius": 47.0949848,
    "std": 33.3248065,
    "ssh": 4.09E-05,
    "weight": 141.51237,
}

# Initialize data structures for features
def initialize_features():
    return {
        "flow_duration": 0,
        "header_length": 0,
        "protocol_type": 0,
        "duration": 0,
        "rate": 0,
        "srate": 0,
        "drate": 0,
        "fin_flag_number": 0,
        "syn_flag_number": 0,
        "rst_flag_number": 0,
        "psh_flag_number": 0,
        "ack_flag_number": 0,
        "ece_flag_number": 0,
        "cwr_flag_number": 0,
        "ack_count": 0,
        "syn_count": 0,
        "fin_count": 0,
        "urg_count": 0,
        "rst_count": 0,
        "http": 0,
        "https": 0,
        "dns": 0,
        "telnet": 0,
        "smtp": 0,
        "ssh": 0,
        "irc": 0,
        "tcp": 0,
        "udp": 0,
        "dhcp": 0,
        "arp": 0,
        "icmp": 0,
```

- Then all the statistical values of the features are calculated using the library Numpy

```
        "dhcp": 0,
        "arp": 0,
        "icmp": 0,
        "ipv": 0,
        "llc": 0,
        "tot_sum": 0.0,
        "min": 0,
        "max": 0,
        "avg": 0,
        "std": 0,
        "tot_size": 0,
        "iat": 0,
        "number": 0,
        "magnitue": 0,
        "radius": 0,
        "covariance": 0,
        "variance": 0,
        "weight": 0,
    }

# Helper function to convert protocol name to number
def protocol_to_number(protocol):
    protocol_map = {
        "icmp": 1,
        "igmp": 2,
        "tcp": 6,
        "udp": 17,
        "ipv6-icmp": 58,
        # Add more protocols if needed
    }
    return protocol_map.get(protocol.lower(), 0)

# Function to calculate header length
def calculate_header_length(log):
    if 'ip' in log:
        return len(log['ip'])
    return 0

# Function to calculate dynamic features
def calculate_dynamic_features(values):
    if len(values) == 0:
        return 0, 0, 0, 0, 0
    magnitue = np.sum(np.abs(values))
```

- All the features are extracted from logs which will be used for calculating statistical and dynamic features as below.

```
    if len(values) == 0:
        return 0, 0, 0, 0, 0
    magnitue = np.sum(np.abs(values))
    radius = np.sqrt(np.sum(np.square(values)))
    covariance = np.cov(values)
    variance = np.var(values)
    weight = np.mean(values)  # Adjust weight calculation as needed
    return magnitue, radius, covariance, variance, weight

# Function to extract features from a single log entry
def extract_features(log):
    features = initialize_features()

    if 'event_type' in log:
        event_type = log['event_type']

        # Handle flow logs
        if event_type == 'flow' or event_type == 'netflow':
            flow_data = log.get('flow', log.get('netflow', {}))
            if 'start' in flow_data and 'end' in flow_data:
                start_time = datetime.strptime(flow_data['start'], "%Y-%m-%dT%H:%M:%S.%f%z")
                end_time = datetime.strptime(flow_data['end'], "%Y-%m-%dT%H:%M:%S.%f%z")
                flow_duration = (end_time - start_time).total_seconds()
                features["flow_duration"] = flow_duration

                protocol = log.get('proto')
                if protocol:
                    features["protocol_type"] = protocol_to_number(protocol)

                bytes_toclient = flow_data.get('bytes_toclient', 0)
                bytes_toserver = flow_data.get('bytes_toserver', 0)
                packets_toclient = flow_data.get('pkts_toclient', 0)
                packets_toserver = flow_data.get('pkts_toserver', 0)
                total_bytes = bytes_toclient + bytes_toserver
                total_packets = packets_toclient + packets_toserver
                duration = flow_duration

                if duration > 0:
                    rate = total_packets / duration
                    srate = packets_toserver / duration
                    drate = packets_toclient / duration
                else:
                    rate, srate, drate = 0, 0, 0
```

```
                    drate = packets_toclient / duration
                else:
                    rate, srate, drate = 0, 0, 0

                features["duration"] = duration
                features["rate"] = rate
                features["srate"] = srate
                features["drate"] = drate
                features["tot_size"] = total_bytes

                # Calculate header length
                header_length = calculate_header_length(log)
                features["header_length"] = header_length

                # Compute inter-arrival times (IAT)
                features["iat"] = duration / total_packets if total_packets > 0 else 0

        # Handle protocol-specific logs
        elif event_type == 'alert':
            proto = log.get('proto', '').lower()
            if proto == 'tcp':
                features["tcp"] += 1
                tcp_flags = log.get('tcp_flags', '')
                if 'F' in tcp_flags:
                    features["fin_flag_number"] += 1
                if 'S' in tcp_flags:
                    features["syn_flag_number"] += 1
                if 'R' in tcp_flags:
                    features["rst_flag_number"] += 1
                if 'P' in tcp_flags:
                    features["psh_flag_number"] += 1
                if 'A' in tcp_flags:
                    features["ack_flag_number"] += 1
                if 'U' in tcp_flags:
                    features["urg_flag_number"] += 1
                if 'E' in tcp_flags:
                    features["ece_flag_number"] += 1
                if 'C' in tcp_flags:
                    features["cwr_flag_number"] += 1

            elif proto == 'http':
                features["http"] += 1
            elif proto == 'https':
```

```
            elif proto == 'http':
                features["http"] += 1
            elif proto == 'https':
                features["https"] += 1
            elif proto == 'dns':
                features["dns"] += 1
            elif proto == 'smtp':
                features["smtp"] += 1
            elif proto == 'ssh':
                features["ssh"] += 1
            elif proto == 'dhcp':
                features["dhcp"] += 1
            elif proto == 'icmp':
                features["icmp"] += 1
            elif proto == 'arp':
                features["arp"] += 1
            elif proto == 'telnet':
                features["telnet"] += 1
            elif proto == 'irc':
                features["irc"] += 1
            elif proto == 'ipv':
                features["ipv"] += 1
            elif proto == 'llc':
                features["llc"] += 1

    # Statistical calculations
    features["tot_sum"] = float(np.sum(features["tot_size"]))
    features["min"] = float(np.min(features["tot_size"])) if features["tot_size"] else 0.0
    features["max"] = float(np.max(features["tot_size"])) if features["tot_size"] else 0.0
    features["avg"] = float(np.mean(features["tot_size"])) if features["tot_size"] else 0.0
    features["std"] = float(np.std(features["tot_size"])) if features["tot_size"] else 0.0

    # Additional derived features
    features["number"] = len([features["tot_size"]])

    # Calculate dynamic features
    features["magnitue"], features["radius"], features["covariance"], features["variance"], f
features([features["tot_size"]]))

    # Convert all NumPy types to native Python types
    for key in features:
        if isinstance(features[key], np.generic):
            features[key] = features[key].item()
```

- Features are passed to model in the order of top 20 for prediction. Along with that time taken and CPU utilisation of ML model as well as Suricata are calculated using psutil library as below.
- By using watchdog, we ensured that the only latest entries of eve.json are parsed.

```
    for key in features:
        if isinstance(features[key], np.generic):
            features[key] = features[key].item()

    # Replace 0 values with mean values
    for key in features:
        if features[key] == 0 and key in mean_values:
            features[key] = mean_values[key]

    # Select only the specified features in the required order
    ordered_features = {
        'iat': features['iat'],
        'magnitue': features['magnitue'],
        'header_length': features['header_length'],
        'rst_count': features['rst_count'],
        'protocol_type': features['protocol_type'],
        'avg': features['avg'],
        'max': features['max'],
        'tot_size': features['tot_size'],
        'urg_count': features['urg_count'],
        'variance': features['variance'],
        'tot_sum': features['tot_sum'],
        'min': features['min'],
        'flow_duration': features['flow_duration'],
        'syn_count': features['syn_count'],
        'srate': features['srate'],
        'rate': features['rate'],
        'radius': features['radius'],
        'std': features['std'],
        'ssh': features['ssh'],
        'weight': features['weight'],
    }

    return ordered_features

# Function to predict and measure time
def predict(features):
    df_features = pd.DataFrame([features])
    start_time = time.time()
    prediction = lgbm_model.predict(df_features)
    prediction_time = time.time() - start_time
    return prediction, prediction_time
```

```
    prediction_time = time.time() - start_time
    return prediction, prediction_time

# Function to get CPU and memory usage
def get_resource_usage(pid):
    process = psutil.Process(pid)
    cpu_usage = process.cpu_percent(interval=1)
    memory_info = process.memory_info()
    memory_usage = memory_info.rss / (1024 * 1024)  # Convert to MB
    return cpu_usage, memory_usage

# Monitor fast.log for alerts
class FastLogHandler(FileSystemEventHandler):
    def __init__(self, eve_json_path):
        self.eve_json_path = eve_json_path
        global initial_eve_json_position
        with open(eve_json_path, 'r') as file:
            file.seek(0, 2)  # Move the cursor to the end of the file
            initial_eve_json_position = file.tell()

    def on_modified(self, event):
        if event.src_path == FAST_LOG_PATH:
            with open(FAST_LOG_PATH, 'r') as fast_log:
                lines = fast_log.readlines()
                if lines:
                    # Trigger prediction on new alerts
                    self.process_alert()

    def process_alert(self):
        global initial_eve_json_position
        with open(self.eve_json_path, 'r') as file:
            file.seek(initial_eve_json_position)
            new_lines = file.readlines()
            initial_eve_json_position = file.tell()
            for line in new_lines:
                log = json.loads(line)
                if log.get('in_iface') in INTERFACES:
                    features = extract_features(log)
                    prediction, prediction_time = predict(features)
                    prediction_label = "Benign" if prediction[0] == "Benign" else "Attack"
                    suricata_cpu, suricata_memory = get_resource_usage(SURICATA_PID)
                    ml_cpu, ml_memory = get_resource_usage(ML_SCRIPT_PID)
                    print(f"Prediction: {prediction_label}, Time taken: {prediction_time} sec
```

```
    def on_modified(self, event):
        if event.src_path == FAST_LOG_PATH:
            with open(FAST_LOG_PATH, 'r') as fast_log:
                lines = fast_log.readlines()
                if lines:
                    # Trigger prediction on new alerts
                    self.process_alert()

    def process_alert(self):
        global initial_eve_json_position
        with open(self.eve_json_path, 'r') as file:
            file.seek(initial_eve_json_position)
            new_lines = file.readlines()
            initial_eve_json_position = file.tell()
            for line in new_lines:
                log = json.loads(line)
                if log.get('in_iface') in INTERFACES:
                    features = extract_features(log)
                    prediction, prediction_time = predict(features)
                    prediction_label = "Benign" if prediction[0] == "Benign" else "Attack"
                    suricata_cpu, suricata_memory = get_resource_usage(SURICATA_PID)
                    ml_cpu, ml_memory = get_resource_usage(ML_SCRIPT_PID)
                    print(f"Prediction: {prediction_label}, Time taken: {prediction_time} seconds")
                    print(f"Suricata CPU: {suricata_cpu}%, Suricata Memory: {suricata_memory} MB")
                    print(f"ML Script CPU: {ml_cpu}%, ML Script Memory: {ml_memory} MB")

# Start monitoring fast.log
def start_monitoring():
    event_handler = FastLogHandler(EVE_JSON_PATH)
    observer = Observer()
    observer.schedule(event_handler, path=FAST_LOG_PATH, recursive=False)
    observer.start()
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()
    observer.join()

# Start monitoring
start_monitoring()
```

10 Testing through attack simulations

- Ensure Kali Linux and smart home network can ping each other. Ideally if all steps are followed as mentioned above it should work.
- Next in Kali Linux install the necessary libraries to perform DDoS attack on mininet devices.

```
pip install hping3
```

- Then by using the below command you can perform the attack on smart home devices.
- TCP flood attack:

sudo hping3 -S --flood -V -p 80 10.0.0.1

- ICMP flood attack:

sudo hping3 --icmp --flood -V 10.0.0.3

Results should show the detection time, CPU usage as well as memory usage as below:

```
Prediction: Attack, Time taken: 0.001219034194946289 seconds
Suricata CPU: 3.0%, Suricata Memory: 873.25 MB
ML Script CPU: 1.0%, ML Script Memory: 159.44140625 MB
```

- DNS tunnelling attack:
- First install iodine in Kali VM

sudo apt-get update

sudo apt-get install iodine

- Similarly install the same on any of the smart home device.

light1 sudo apt-get install iodine

- Create a DNS server in kali VM

sudo iodined -f -c -P mypassword <IP of Kali> tunnel.example.com

- Again, login to the same device where iodine is installed and run the command below:

light1 sudo iodine -f -P mypassword <kali ip> tunnel.example.com

- Now the tunnel is established, which can be viewed using netcat server of kali.

nc <mininet device ip> 8080

- By just creating a tunnel the Suricata will alert the DNS tunnelling and also the script should detect the attack and provide the time taken for detection as well as CPU usage details as below.

```
Prediction: Attack, Time taken: 0.0056536197662353516 seconds
```

```
  2848 root       20   0 1744M  837M  7972 S  0.7 21.4  0:00.22 suricata -c /et
  3133 navya      20   0  631M  159M 53396 S  0.7  4.1  0:01.39 python3 pythonm
```

- Mirai botnet attack was not directly conducted on smart home network, rather pcap file of Mirai attack traffic was collected and necessary network features were extracted into a csv file. For this download the file from https://mcfp.felk.cvut.cz/publicDatasets/IoTDatasets/CTU-IoT-Malware-Capture-34-1/ (Stratosphere IPS, 2023).

```python
import pyshark
import pandas as pd
import numpy as np
import nest_asyncio
import asyncio

nest_asyncio.apply()

async def extract_features_from_pcap(pcap_file):
    # Read pcap file
    cap = pyshark.FileCapture(pcap_file)

    # Initialize lists to store extracted features
    features = []

    for packet in cap:
        if 'IP' in packet:
            pkt_features = {}
            ip_layer = packet['IP']
            transport_layer = packet.transport_layer

            # Basic packet information
            pkt_features['flow_duration'] = float(packet.sniff_time.timestamp())
            pkt_features['header_length'] = int(ip_layer.hdr_len)
            pkt_features['protocol_type'] = ip_layer.proto
            pkt_features['duration'] = float(packet.sniff_time.timestamp())

            # Calculate rate (packet size over duration)
            pkt_features['tot_size'] = int(ip_layer.len)
            pkt_features['rate'] = pkt_features['tot_size'] / pkt_features['duration']
```

```python
            # Calculate srate and drate if previous packet exists
            if len(features) > 0:
                prev_pkt = features[-1]
                pkt_features['srate'] = prev_pkt['tot_size'] / prev_pkt['duration']
                pkt_features['drate'] = (pkt_features['tot_size'] - prev_pkt['tot_size']) / (pkt_features['duration'] - prev_pkt['duration'])
            else:
                pkt_features['srate'] = 0
                pkt_features['drate'] = 0

            # Flags (set to 0 if not a TCP packet)
            if transport_layer == 'TCP':
                pkt_features['fin_flag_number'] = int(getattr(packet.tcp, 'flags_fin', 0))
                pkt_features['syn_flag_number'] = int(getattr(packet.tcp, 'flags_syn', 0))
                pkt_features['rst_flag_number'] = int(getattr(packet.tcp, 'flags_rst', 0))
                pkt_features['psh_flag_number'] = int(getattr(packet.tcp, 'flags_psh', 0))
                pkt_features['ack_flag_number'] = int(getattr(packet.tcp, 'flags_ack', 0))
                pkt_features['ece_flag_number'] = int(getattr(packet.tcp, 'flags_ece', 0))
                pkt_features['cwr_flag_number'] = int(getattr(packet.tcp, 'flags_cwr', 0))
            else:
                pkt_features['fin_flag_number'] = 0
                pkt_features['syn_flag_number'] = 0
                pkt_features['rst_flag_number'] = 0
                pkt_features['psh_flag_number'] = 0
                pkt_features['ack_flag_number'] = 0
                pkt_features['ece_flag_number'] = 0
                pkt_features['cwr_flag_number'] = 0

            # Counts
            pkt_features['ack_count'] = pkt_features['ack_flag_number']
            pkt_features['syn_count'] = pkt_features['syn_flag_number']
            pkt_features['fin_count'] = pkt_features['fin_flag_number']
            pkt_features['urg_count'] = 0
            pkt_features['rst_count'] = pkt_features['rst_flag_number']
```

```python
            # Protocols (check if the protocol is being used)
            pkt_features['http'] = 1 if 'HTTP' in packet else 0
            pkt_features['https'] = 1 if 'HTTPS' in packet else 0
            pkt_features['dns'] = 1 if 'DNS' in packet else 0
            pkt_features['telnet'] = 1 if 'TELNET' in packet else 0
            pkt_features['smtp'] = 1 if 'SMTP' in packet else 0
            pkt_features['ssh'] = 1 if 'SSH' in packet else 0
            pkt_features['irc'] = 1 if 'IRC' in packet else 0
            pkt_features['tcp'] = 1 if transport_layer == 'TCP' else 0
            pkt_features['udp'] = 1 if transport_layer == 'UDP' else 0
            pkt_features['dhcp'] = 1 if 'DHCP' in packet else 0
            pkt_features['arp'] = 1 if 'ARP' in packet else 0
            pkt_features['icmp'] = 1 if 'ICMP' in packet else 0
            pkt_features['ipv'] = 1 if 'IPV' in packet else 0
            pkt_features['llc'] = 0  # Not typically found in pcap files, set to 0

            # Append extracted features to the list
            features.append(pkt_features)

    cap.close()

    # Convert list to DataFrame
    df = pd.DataFrame(features)

    # Calculate statistical features
    df['tot_sum'] = df['tot_size'].sum()
    df['min'] = df['tot_size'].min()
    df['max'] = df['tot_size'].max()
    df['avg'] = df['tot_size'].mean()
    df['std'] = df['tot_size'].std()
```

```python
    # Calculate inter-arrival times (iat) and related statistical features
    df['iat'] = df['flow_duration'].diff().fillna(0)
    df['number'] = df['iat'].count()
    df['magnitude'] = df['iat'].abs().sum()
    df['radius'] = np.sqrt((df['iat'] ** 2).sum())
    df['covariance'] = df['iat'].cov(df['tot_size'])
    df['variance'] = df['iat'].var()
    df['weight'] = df['tot_size'].sum()

    # Assign a label (assuming binary classification: 0 for normal, 1 for attack)
    df['label'] = 1  # Assuming all packets in this pcap are part of an attack

    # Save DataFrame to CSV
    df.to_csv('extracted_features.csv', index=False)

    return df

# Path to your pcap file
pcap_file = '/path/to/pcap/2018-12-21-15-50-14-192.168.1.195.pcap'

# Extract features
features_df = asyncio.run(extract_features_from_pcap(pcap_file))
```

- Then this file was used as input for saved model for prediction. The attack traffic was predicted correctly by the model as in below screenshots and during which the time for prediction on each sample as well as the CPU utilisation is calculated.

```
Single prediction time: 0.006418 seconds
CPU usage for prediction: 28.30%
Memory usage for prediction: 0.03 MB
```

- The output shows as below where we can see that Mirai attack as well as other network traffic attacks were predicted.

std	iat	number	magnitude	radius	covariance	variance	weight	label
612.0909633	0	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	BruteForce
612.0909633	0.00499916	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	BruteForce
612.0909633	0.00175285	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	Recon
612.0909633	0.08069396	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	BruteForce
612.0909633	0.00024915	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	Recon
612.0909633	0.00075102	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	BruteForce
612.0909633	0.00024891	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	Recon
612.0909633	0.00150108	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	BruteForce
612.0909633	7.867813110	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	Recon
612.0909633	0.00099015	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	Recon
612.0909633	1.059396982	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	Recon
612.0909633	2.079813957	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	Recon
612.0909633	4.079935073	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	Recon
612.0909633	8.160090923	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	Recon
612.0909633	8.117368936	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	DoS
612.0909633	0.00149608	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	Mirai
612.0909633	8.52114201	228469	86397.07132	1541.90778	-166.06847	10.26317902	118451876	Recon
612.0909633	0.01673889	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	Recon
612.0909633	0.00074911	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	Recon
612.0909633	0.00049996	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	Recon
612.0909633	0.00074911	228469	86397.07132	1541.907783	-166.06847	10.26317902	118451876	BruteForce

- Below is the script used for this prediction and evaluation of Mirai attack.

```python
import pandas as pd
import joblib
import time
import psutil

# Load the model from the file
model_filename = 'lgbm_model_top_features_regularized.pkl'
loaded_model = joblib.load(model_filename)

# Load the new CSV file with extracted features
new_csv_path = 'extracted_features.csv'
new_data = pd.read_csv(new_csv_path)

# Filter the DataFrame to include only the top 20 features
top_20_features = ['iat', 'magnitude', 'header_length', 'rst_count', 'protocol_type', 'avg',
                   'max', 'tot_size', 'urg_count', 'variance', 'tot_sum', 'min',
                   'flow_duration', 'syn_count', 'srate', 'rate', 'radius', 'std', 'ssh',
                   'weight']
new_data_top_features = new_data[top_20_features]

# Measure time and resource usage for a single prediction
example_data = new_data_top_features.iloc[0:1]

# Get the initial CPU and memory usage
initial_cpu_percent = psutil.cpu_percent(interval=None)
initial_memory_info = psutil.virtual_memory()

# Measure the time taken for the prediction
start_time = time.time()
prediction = loaded_model.predict(example_data)
end_time = time.time()
elapsed_time = end_time - start_time
```

```python
# Measure the time taken for the prediction
start_time = time.time()
prediction = loaded_model.predict(example_data)
end_time = time.time()
elapsed_time = end_time - start_time

# Get the final CPU and memory usage
final_cpu_percent = psutil.cpu_percent(interval=None)
final_memory_info = psutil.virtual_memory()

cpu_usage = final_cpu_percent - initial_cpu_percent
memory_usage = initial_memory_info.used - final_memory_info.used

print(f"Single prediction time: {elapsed_time:.6f} seconds")
print(f"CPU usage for prediction: {cpu_usage:.2f}%")
print(f"Memory usage for prediction: {memory_usage / (1024 * 1024):.2f} MB")

# Make predictions using the loaded model for the entire dataset
predictions = loaded_model.predict(new_data_top_features)

# Add the predictions to the DataFrame as a new column 'label'
new_data['label'] = predictions

# Save the DataFrame with predictions to a new CSV file
output_csv_path = 'new_pcap_features_with_predictions.csv'
new_data.to_csv(output_csv_path, index=False)

print(f"Predictions saved to {output_csv_path}")
```

```
[LightGBM] [Warning] bagging_freq is set=1, subsample_freq=0 will be ignored. Current value: bagging_freq=1
[LightGBM] [Warning] feature_fraction is set=0.8, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.8
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored. Current value: bagging_fraction=0.8
Single prediction time: 0.006418 seconds
CPU usage for prediction: 28.30%
Memory usage for prediction: 0.03 MB
```

References

docs.suricata.io. (2024). *3. Installation — Suricata 7.0.2-dev documentation*. Available at: https://docs.suricata.io/en/latest/install.html [Accessed 6 August 2024]

Mininet Team (2018). *Mininet Walkthrough - Mininet*. Available at: http://mininet.org/walkthrough/ [Accessed 6 August 2024]

Neto, E.C.P., Dadkhah, S., Ferreira, R., Zohourian, A., Lu, R. and Ghorbani, A.A. (2023) 'CICIoT2023: A real-time dataset and benchmark for large-scale attacks in IoT environment', *Sensors*, 23(13), pp.5941. doi: 10.3390/s23135941

nikitastsinnas (2024). *IDS-8*. Available at: https://www.kaggle.com/code/nikitastsinnas/ids-8 [Accessed 6 August 2024].

research.google.com. (2023) . Google Colab. Available at: https://research.google.com/colaboratory/faq.html#:~:text=Colab%20is%20a%20hosted%20Jupyter [Accessed 13 July 2024]

Stratosphere IPS. (2023) *Aposemat Project: IoT Malware Datasets*. Available at: https://www.stratosphereips.org/datasets-iot [Accessed 22 Jul. 2024].