INDIAN INSTITUTE OF TECHNOLOGY

BANARAS HINDU UNIVERSITY, VARANASI

CSE-530: Information Security, Project Report

Wireless Intrusion Detection System

1. Objective

To develop, train and test an Intrusion Detection System for modern wireless based attacks.

2. Problem

2.1 Problem Description

Wireless Networks facilitate the ease of communication for sharing the crucial information. Recently, most of the small and large-scale companies, educational institutions, government organizations, medical sectors, military and banking sectors are using the wireless networks. Security threats, a common term found both in wired as well as in wireless networks. However, it holds lot of importance in wireless networks because of its susceptible nature to threats. Security concerns in WLAN are studied and many organizations concluded that Wireless Intrusion Detection Systems (WIDS) is an essential element in network security infrastructure to monitor wireless activity for signs of attacks. However, it is an indisputable fact that the art of detecting attacks remains in its infancy.

For training and testing a WIDS, we make use of publicly available (on request) datasets such as Aegean Wi-Fi Intrusion Dataset (AWID). So, the problem reduces to a machine / deep learning multi-class classification problem, that is, given a packet or frame we classify it into one of the attack classes or a normal packet.

2.2 Problem Constraints

1. Cost of misclassification is high.
2. Latency should be low.
3. Interpretability is nice to have.

2.3 Problem Metrics

1. Cost of misclassification of an attack packet is high, that is, we do not want any false negative classifications.
2. High recall scores will ensure that.

3. Objective as Deep Learning problem and implementation details

Two models, one to classify traditional attacks and other for modern attacks, are trained on three different datasets, which are, AWID-2, AWID-3 and self-created dataset.

3.1 Data Overview:

- Dataset: Subset of AWID-2
- Format: CSV
- Size: 1.04 GB
- Data points: 2371216
- Features: 155
- Classes: 17

- Dataset: Subset of AWID-3
- Format: PCAP (Converted to CSV)
- Size: 241 MB
- Data points: 1006568
- Features: 35 (from 1000s)
- Classes: 13

- Dataset: Self-Created
- Format: PCAP and CSV

- Size: 19.5 MB
- Data points: 151562
- Features: 24
- Classes: 5

3.2 Attack Classes

3.2.1 AWID-2

1. ARP (Address Resolution Protocol): Associate the attacker's MAC address with the IP address of another host, such as the default gateway, causing any traffic meant for that IP address to be sent to the attacker instead.
2. Authentication: Flooding attack attempts to exhaust the physical resources of the APs.
3. AMOK: Flooding attack, like deauthentication
4. Beacon: Attacker fakes these beacon frames and sends them in large numbers to confuse the wireless clients.
5. Cafe-Latte: Obtain a WEP key from a client system, done by capturing an ARP packet from the client, manipulating it and sending it back to client, who in turn generates packets used to determine WEP key.
6. Chop-Chop: Works by taking one byte of data from a WEP encrypted packet, substituting values for that byte, and recalculating the encryption checksum.
7. RTS/CTS attack: is a type of low-rate DoS attacks. RTS/CTS (Request to Send / Clear to Send) mechanism is a reservation scheme used in the wireless networks
8. Deauthentication: It is a DoS attack, that sends disassociate packets to one or more clients which are currently associated with a particular access point.
9. Disassociation: Targeted deauthentication attack
10. Evil-Twin: A fraudulent Wi-Fi access point that appears to be legitimate but is set up to eavesdrop on wireless communications.
11. Fragmentation: Aims at revealing a significant portion of the keystream by sending notably less messages than the ChopChop. The keystream can later be used to generate and inject packets into the network as part of other assaults.

12. Hirte: It is another method for retrieving the WEP key using solely a client and not needing an AP of the network at all. It works in a similar fashion to the Caffe Latte, but it incorporates methods found in the fragmentation attack.
13. Probe Request: Flooding attack aims at stressing the resources of an AP and eventually drive it to paralysis.
14. Probe Response: This flooding attack also takes advantage of the Probe mechanism, it works in reverse by targeting the client rather than the AP.
15. Power Saving: DoS attack, it basically tricks the AP into thinking that a specific STA has fallen into doze mode.

3.2.2 AWID-3

1. Association
2. Deauthentication
3. Evil-Twin
4. Krack: It takes advantage of the fact that retransmissions of either the first or the third message of the 4-way handshake should occur if the AP does not receive message two or four, respectively. Based on this, the attacker acting as a man-in-the-middle (MitM), blocks message four from reaching the AP, thus making the AP willingly re-transmit the same message toward the STA. This would induce a reinstallation of the same Pairwise Transient Key (PTK) / Group Transient Key (GTK) / Integrity Group Temporal Key (IGTK) at the STA side, meaning that as the STA sends its next data frame, the data WPA2 protocol will reuse the nonces transmitted during the original 4-way handshake. By forcing nonce reuse in this manner, the encryption protocol can be attacked, e.g., packets can be replayed, decrypted, and/or forged.
5. Krook: It is related to Krack, applies to WPA2 as well, but the attacker is not required to be authenticated and associated to the wireless network. The vulnerability is rooted in the fact that as soon as an STA is disassociated, the TK is cleared in memory, i.e., set to all-zero. While this is normal, it was observed that all data frames left in the Wireless Network Controller's (WNIC) egress queue might be then transmitted while encapsulated with this static all-zero key.
6. Rogue AP: Rogue access points that are installed by employees (malicious users or by mistake). In the situation where they are misconfigured or configured without any security, it opens a next attack surface for having easy access to a very secure network

7. Spoofing: The assailant clones the front webpage of a popular website, say, Instagram. Then, they perform ARP and DNS spoofing to redirect the users to their fake webpage instead of the original one.

3.3.3 Self-Created

1. Association Request
2. Association Response
3. Authentication
4. Deauthentication

3.4 High level explanation

For all the three datasets, the following was performed in a sequence:

1. Data Loading
2. Exploratory data analysis: Univariate, multivariate analysis was performed by plotting graphs and visualizing deviations and how much of the classes are separable by using single or a combination of features. This gives insight about feature importance like the wlan_fc_type (control frame) and related types are highly important.
3. Cleaning: Column which have a huge number of NULL values (>80%) were dropped, same with the rows and also column having zero standard deviation were also removed.
4. Feature selection: After gaining some domain knowledge and analyzing features, a few subsets of features were finally chosen for the classification.
5. Pre-processing: NULL values were imputed using mean values, numerical features were standardized and categorical features were encoded using Bag of Words (One Hot Encoding) method.
6. Up-sampling was performed using SMOTE (Simple Minority Sampling Technique), which creates new data points around the existing one and in the path connecting two data points.
7. Then data is split into train (66%), validation (~22%) and test (~11%) sets.

3.5 Model Implementation Details and Results

Now, for the two datasets AWID-2 and AWID-3, two different model iterations were used. For the AWID-2 dataset, a simple sequential full-connected four layered model was used which performed well on all three splits with ~99% accuracy. But while testing it on a completely different distribution of data, the model did not do well in accuracy score which is expected but still achieved 98% recall which is what is required. The low accuracy is due to the various factors like version of wire-shark used to capture data, monitor node, background noise packet and more, which is why a model trained on one distributed should not be used for another. Which is why for live testing a new dataset was created.

For AWID-3, which contains complex attacks like Krack, Krook, RougeAP etc, these attacks have a pattern and are harder to detect, so a layer which extracts time sequence information was used which is GRU (Gated Recurrent Unit), along with the dense layers. This layer expects history data points along with the current point, so the data-frame was modified so that every point also has some history points paired with it. This type of model turned out to be immensely successful with a 100% accuracy score for all attacks with no help of artificial points (upsampling).

3.6 Deployed Model Testing

Since, the results were good, a new dataset was created with a few attack classes. It was labelled using frame time and attack start time values as reference, just like how AWID-3 was labelled while converting from PCAP to CSV.

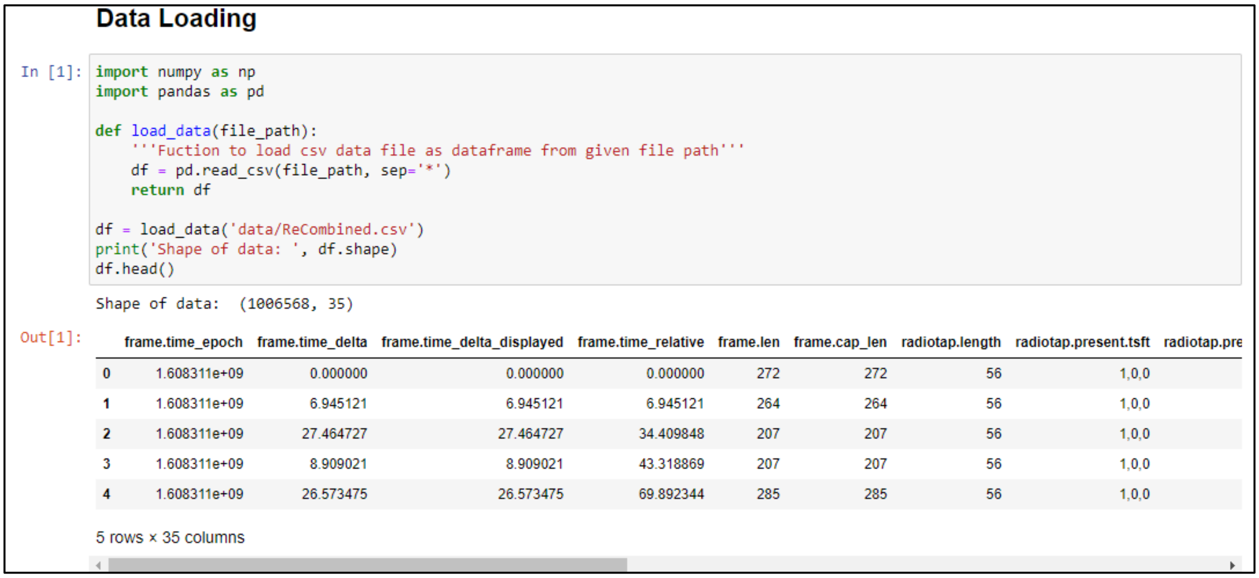
The dataset was preprocessed and simple model was trained on the dataset.

For testing the model, a sub process on a different thread captured packets using tshark, another thread for cleaning each packet and running the model on it, and another for the GUI class for prediction speed control and normal packet show flag.

The attacks were executed using pen-testing tools from Kali Linux and an external wireless adapter node was used for packet capture and attack injection.

4. Screenshots

[AWID-3]



Data Loading

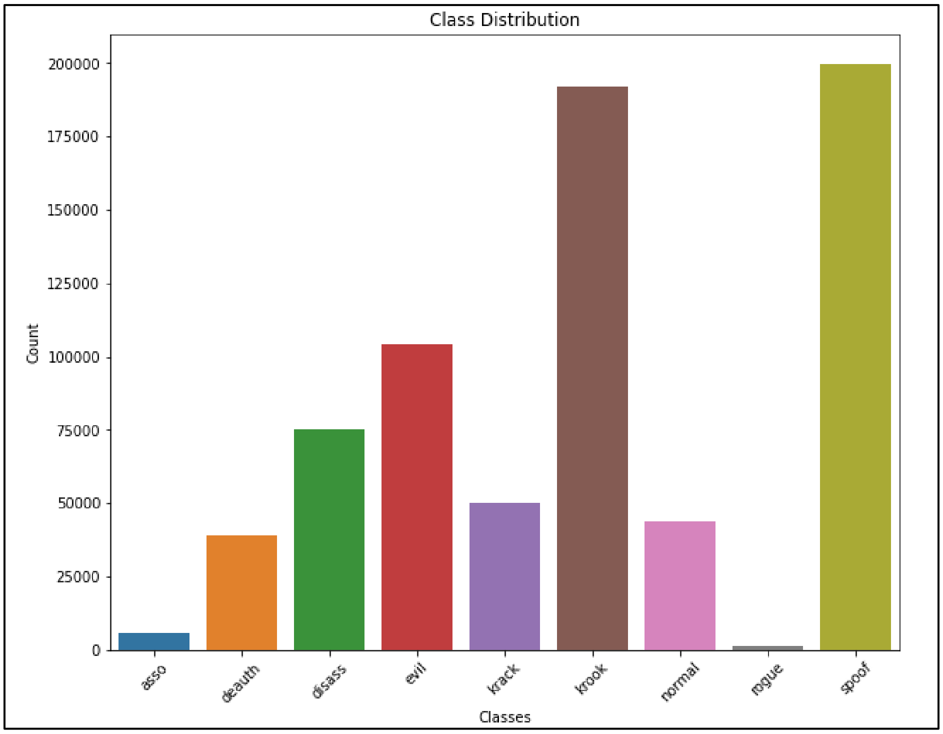
In [1]:
```
import numpy as np
import pandas as pd

def load_data(file_path):
    '''Fuction to load csv data file as dataframe from given file path'''
    df = pd.read_csv(file_path, sep='*')
    return df

df = load_data('data/ReCombined.csv')
print('Shape of data: ', df.shape)
df.head()
```

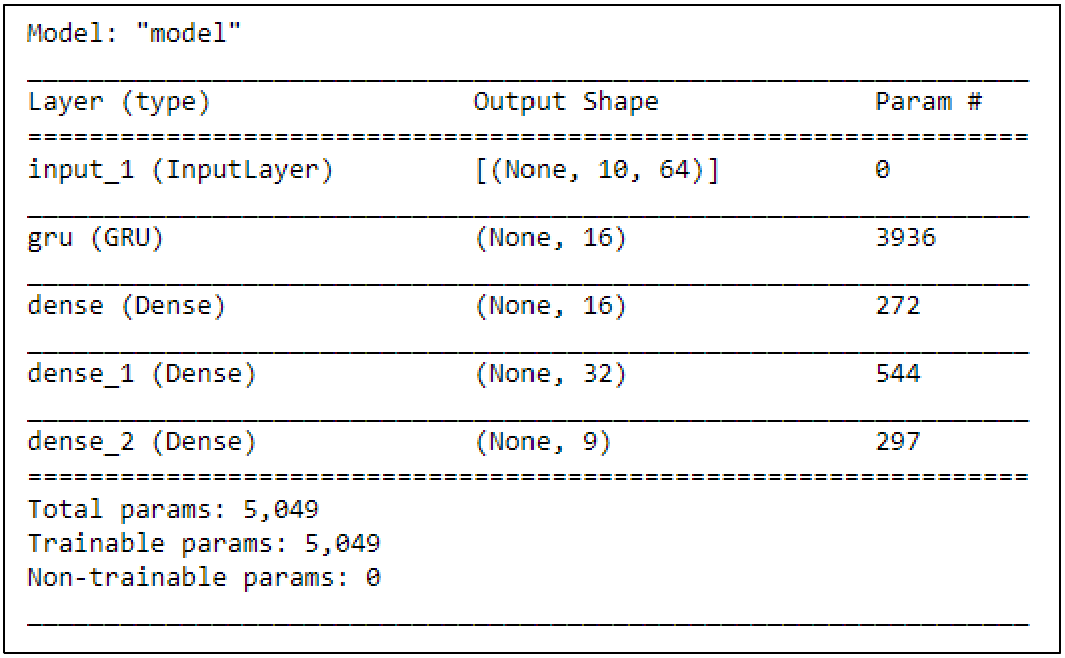
```
Shape of data:  (1006568, 35)
```

Out[1]:

	frame.time_epoch	frame.time_delta	frame.time_delta_displayed	frame.time_relative	frame.len	frame.cap_len	radiotap.length	radiotap.present.tsft	radiotap.pre
0	1.608311e+09	0.000000	0.000000	0.000000	272	272	56	1,0,0	
1	1.608311e+09	6.945121	6.945121	6.945121	264	264	56	1,0,0	
2	1.608311e+09	27.464727	27.464727	34.409848	207	207	56	1,0,0	
3	1.608311e+09	8.909021	8.909021	43.318869	207	207	56	1,0,0	
4	1.608311e+09	26.573475	26.573475	69.892344	285	285	56	1,0,0	

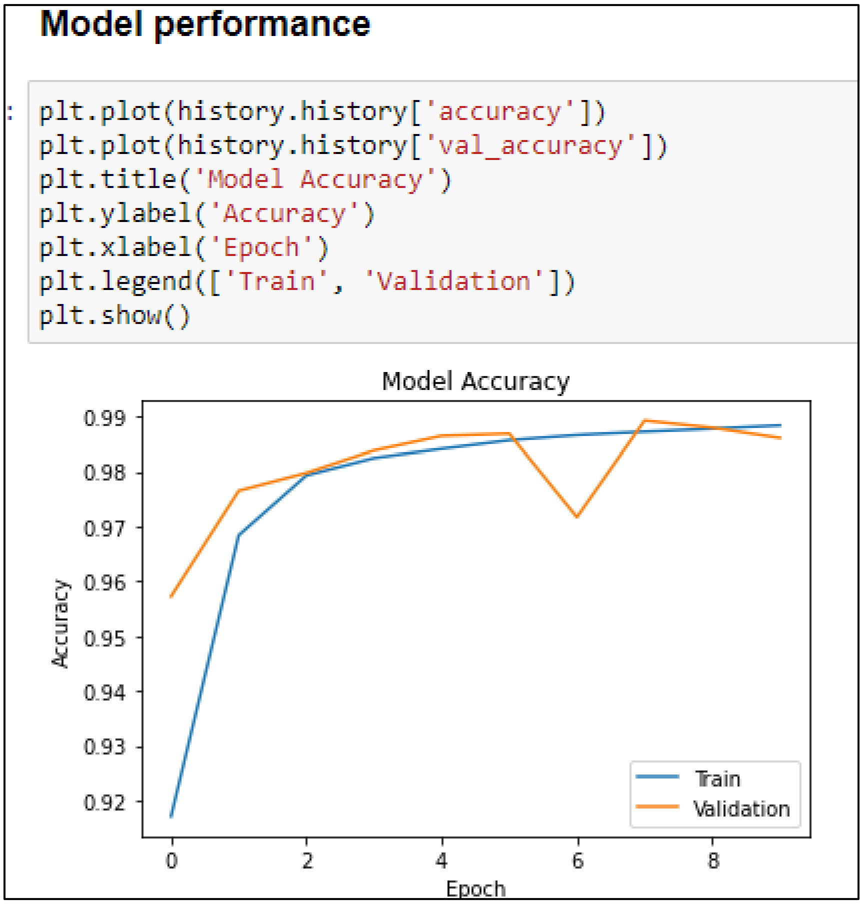
5 rows × 35 columns





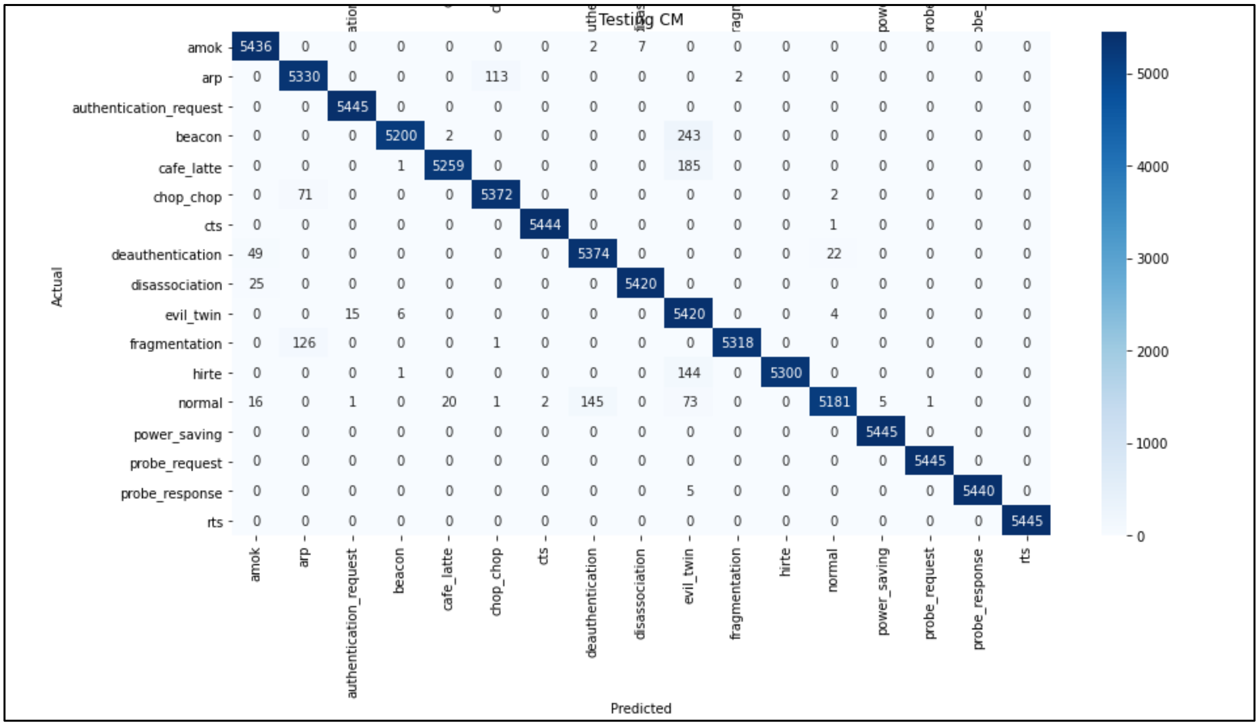
```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 10, 64)]          0         
_________________________________________________________________
gru (GRU)                    (None, 16)                3936      
_________________________________________________________________
dense (Dense)                (None, 16)                272       
_________________________________________________________________
dense_1 (Dense)              (None, 32)                544       
_________________________________________________________________
dense_2 (Dense)              (None, 9)                 297       
=================================================================
Total params: 5,049
Trainable params: 5,049
Non-trainable params: 0
_________________________________________________________________
```

[AWID-2] Results:

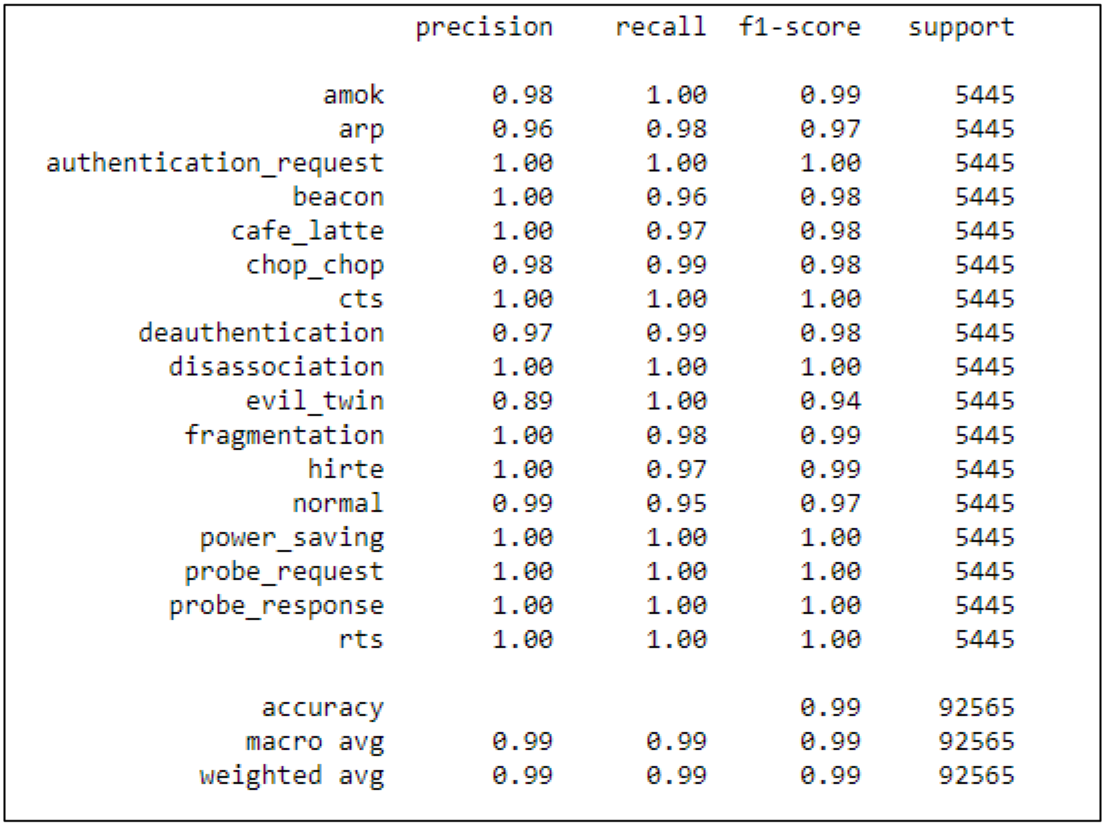


Model performance

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'])
plt.show()
```

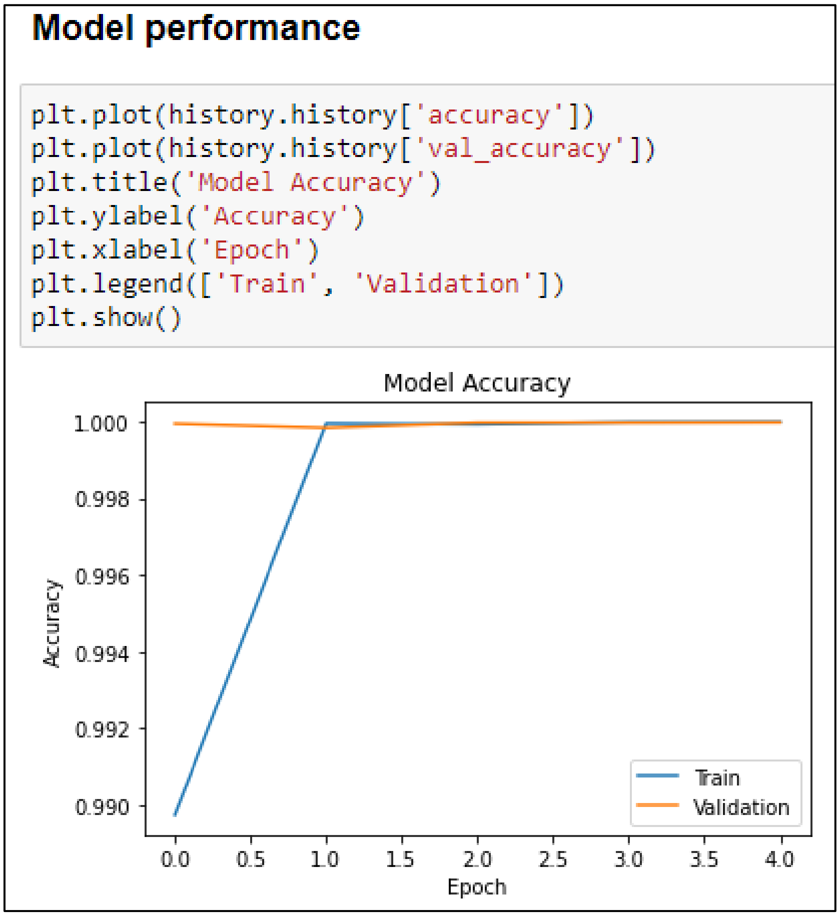


	amok	arp	authentication_request	beacon	cafe_latte	chop_chop	cts	deauthentication	disassociation	evil_twin	fragmentation	hirte	normal	power_saving	probe_request	probe_response	rts
amok	5436	0	0	0	0	0	0	2	7	0	0	0	0	0	0	0	0
arp	0	5330	0	0	0	113	0	0	0	0	2	0	0	0	0	0	0
authentication_request	0	0	5445	0	0	0	0	0	0	0	0	0	0	0	0	0	0
beacon	0	0	0	5200	2	0	0	0	0	243	0	0	0	0	0	0	0
cafe_latte	0	0	0	1	5259	0	0	0	0	185	0	0	0	0	0	0	0
chop_chop	0	71	0	0	0	5372	0	0	0	0	0	0	2	0	0	0	0
cts	0	0	0	0	0	0	5444	0	0	0	0	0	1	0	0	0	0
deauthentication	49	0	0	0	0	0	0	5374	0	0	0	0	22	0	0	0	0
disassociation	25	0	0	0	0	0	0	0	5420	0	0	0	0	0	0	0	0
evil_twin	0	0	15	6	0	0	0	0	0	5420	0	0	4	0	0	0	0
fragmentation	0	126	0	0	0	1	0	0	0	0	5318	0	0	0	0	0	0
hirte	0	0	0	1	0	0	0	0	0	144	0	5300	0	0	0	0	0
normal	16	0	1	0	20	1	2	145	0	73	0	0	5181	5	1	0	0
power_saving	0	0	0	0	0	0	0	0	0	0	0	0	0	5445	0	0	0
probe_request	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5445	0	0
probe_response	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	5440	0
rts	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5445



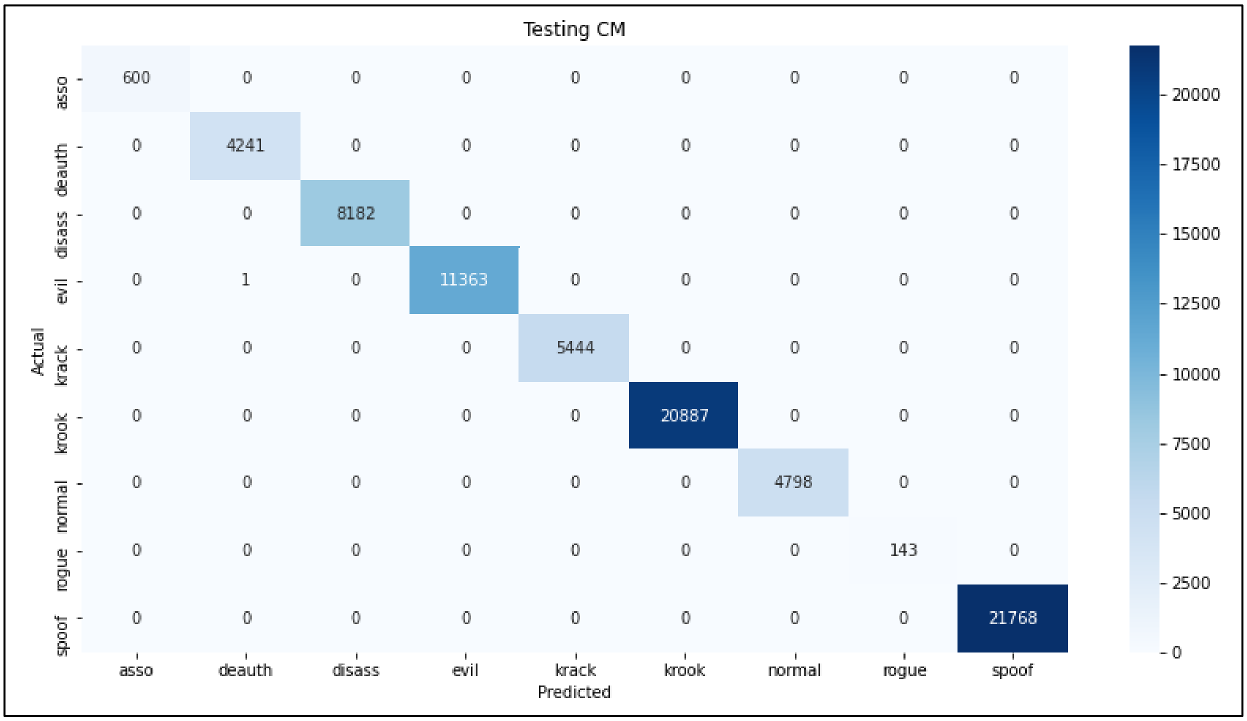
```
                        precision    recall  f1-score   support

                  amok       0.98      1.00      0.99      5445
                   arp       0.96      0.98      0.97      5445
authentication_request       1.00      1.00      1.00      5445
                beacon       1.00      0.96      0.98      5445
            cafe_latte       1.00      0.97      0.98      5445
             chop_chop       0.98      0.99      0.98      5445
                   cts       1.00      1.00      1.00      5445
      deauthentication       0.97      0.99      0.98      5445
        disassociation       1.00      1.00      1.00      5445
             evil_twin       0.89      1.00      0.94      5445
         fragmentation       1.00      0.98      0.99      5445
                 hirte       1.00      0.97      0.99      5445
                normal       0.99      0.95      0.97      5445
          power_saving       1.00      1.00      1.00      5445
         probe_request       1.00      1.00      1.00      5445
        probe_response       1.00      1.00      1.00      5445
                   rts       1.00      1.00      1.00      5445

              accuracy                           0.99     92565
             macro avg       0.99      0.99      0.99     92565
          weighted avg       0.99      0.99      0.99     92565
```

[AWID-3] Results:

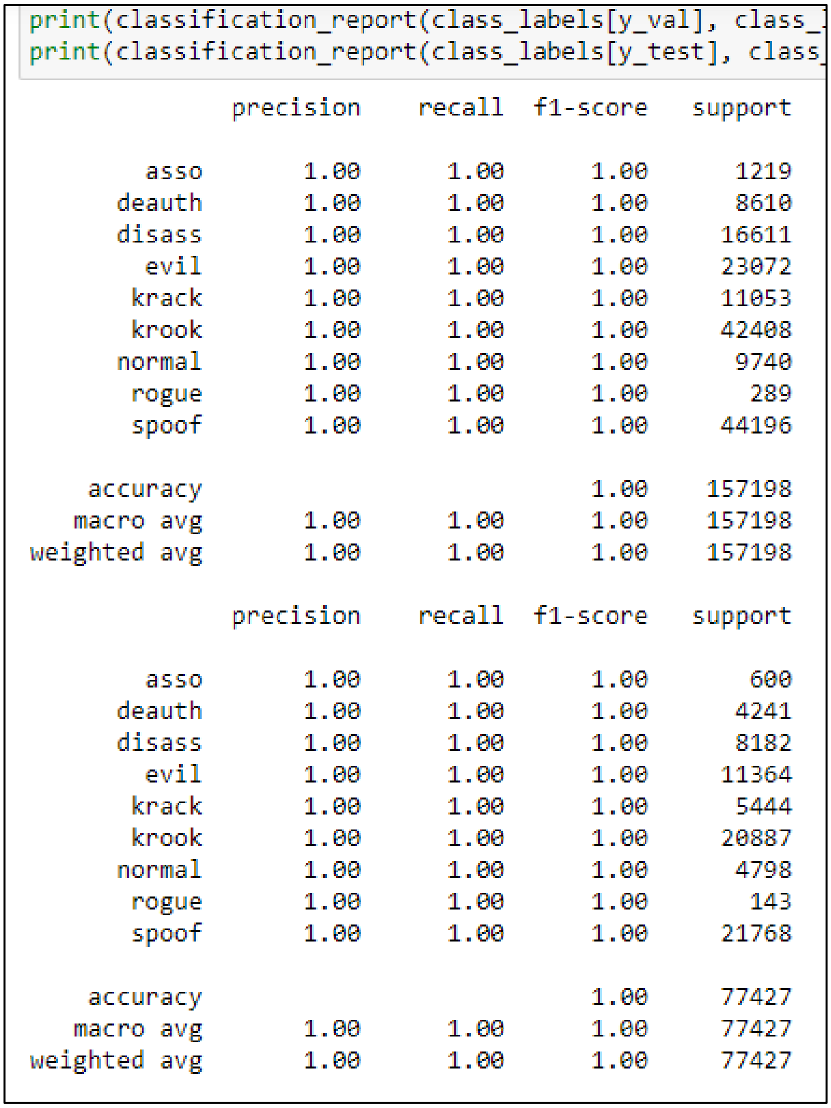


Model performance

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'])
plt.show()
```



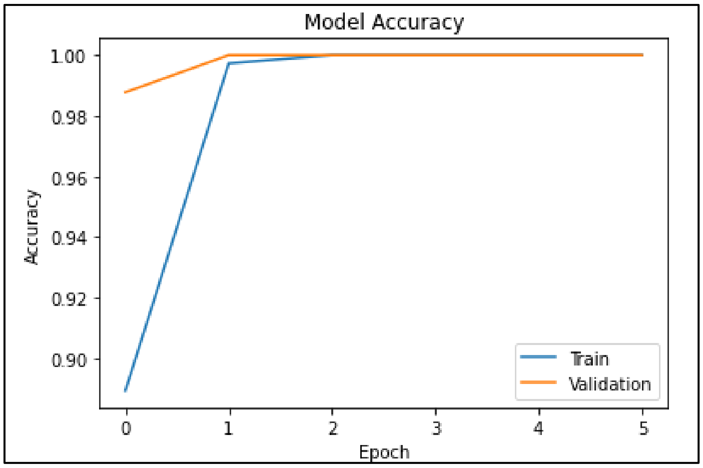
	asso	deauth	disass	evil	krack	krook	normal	rogue	spoof
asso	600	0	0	0	0	0	0	0	0
deauth	0	4241	0	0	0	0	0	0	0
disass	0	0	8182	0	0	0	0	0	0
evil	0	1	0	11363	0	0	0	0	0
krack	0	0	0	0	5444	0	0	0	0
krook	0	0	0	0	0	20887	0	0	0
normal	0	0	0	0	0	0	4798	0	0
rogue	0	0	0	0	0	0	0	143	0
spoof	0	0	0	0	0	0	0	0	21768

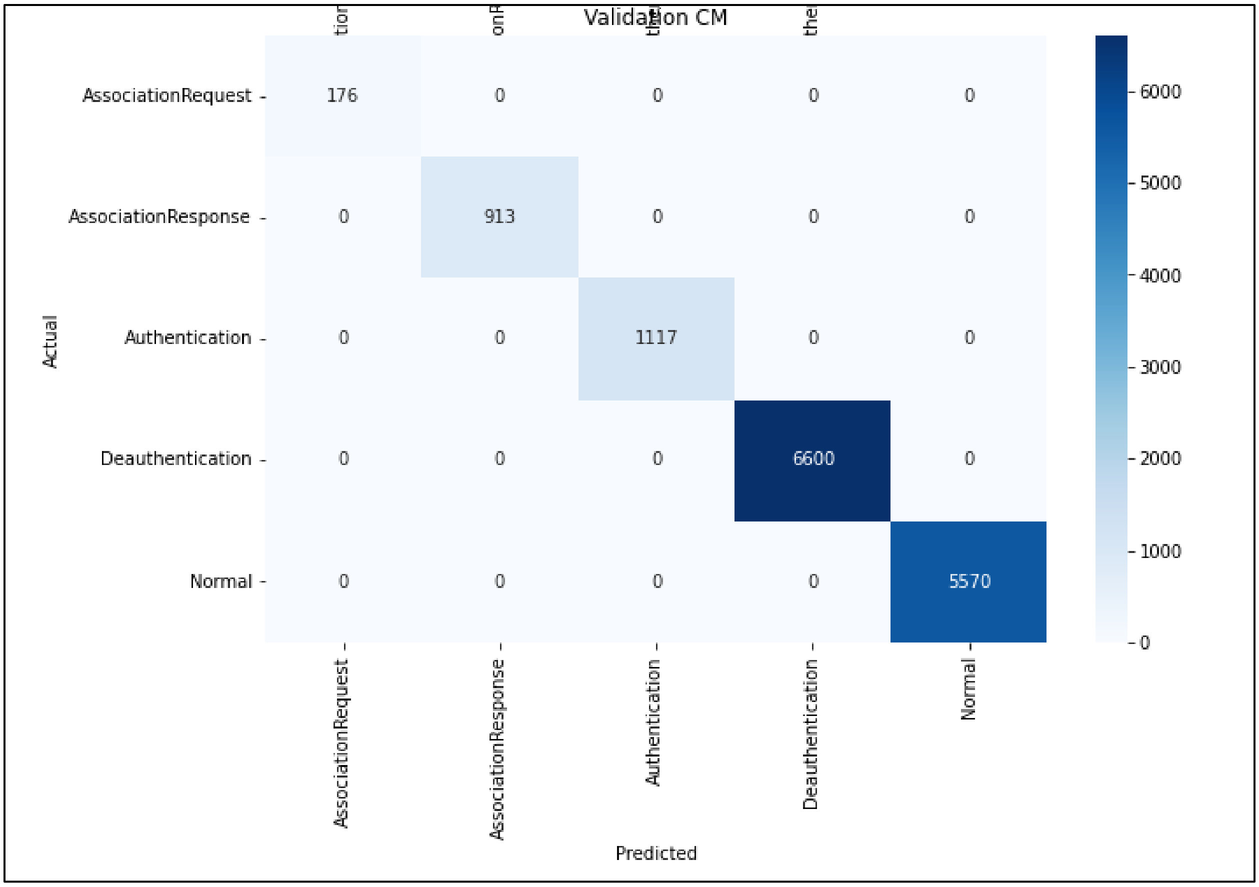


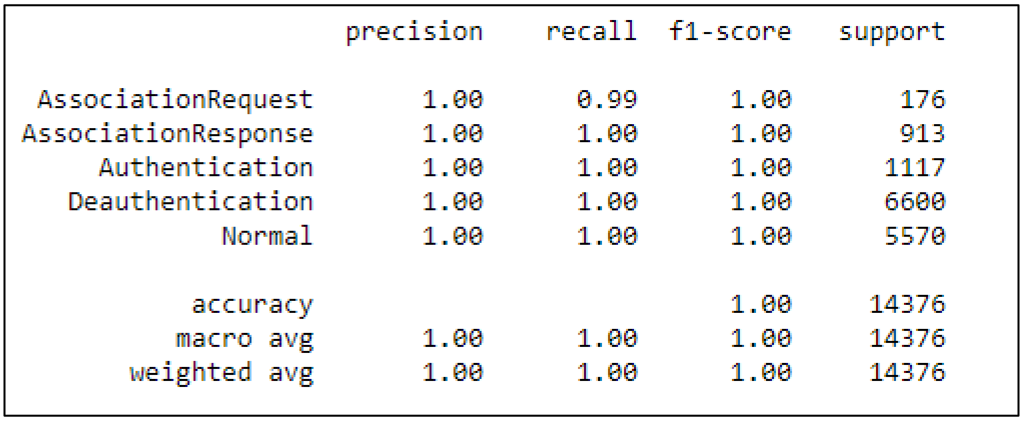
```
print(classification_report(class_labels[y_val], class_
print(classification_report(class_labels[y_test], class_
```

```
              precision    recall  f1-score   support

        asso       1.00      1.00      1.00      1219
      deauth       1.00      1.00      1.00      8610
      disass       1.00      1.00      1.00     16611
        evil       1.00      1.00      1.00     23072
       krack       1.00      1.00      1.00     11053
       krook       1.00      1.00      1.00     42408
      normal       1.00      1.00      1.00      9740
       rogue       1.00      1.00      1.00       289
       spoof       1.00      1.00      1.00     44196

    accuracy                           1.00    157198
   macro avg       1.00      1.00      1.00    157198
weighted avg       1.00      1.00      1.00    157198

              precision    recall  f1-score   support

        asso       1.00      1.00      1.00       600
      deauth       1.00      1.00      1.00      4241
      disass       1.00      1.00      1.00      8182
        evil       1.00      1.00      1.00     11364
       krack       1.00      1.00      1.00      5444
       krook       1.00      1.00      1.00     20887
      normal       1.00      1.00      1.00      4798
       rogue       1.00      1.00      1.00       143
       spoof       1.00      1.00      1.00     21768

    accuracy                           1.00     77427
   macro avg       1.00      1.00      1.00     77427
weighted avg       1.00      1.00      1.00     77427
```

[Self-Created] Results:

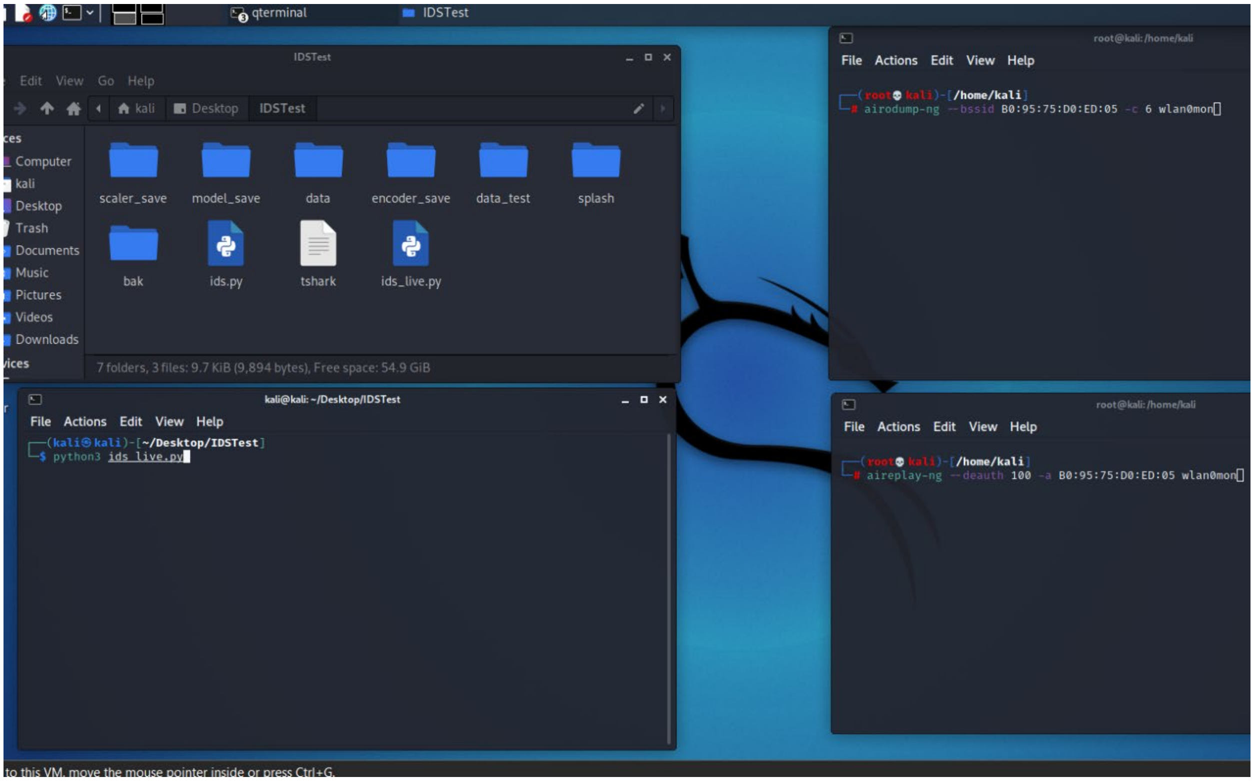






```
                     precision    recall  f1-score   support

  AssociationRequest       1.00      0.99      1.00       176
 AssociationResponse       1.00      1.00      1.00       913
      Authentication       1.00      1.00      1.00      1117
    Deauthentication       1.00      1.00      1.00      6600
              Normal       1.00      1.00      1.00      5570

            accuracy                           1.00     14376
           macro avg       1.00      1.00      1.00     14376
        weighted avg       1.00      1.00      1.00     14376
```

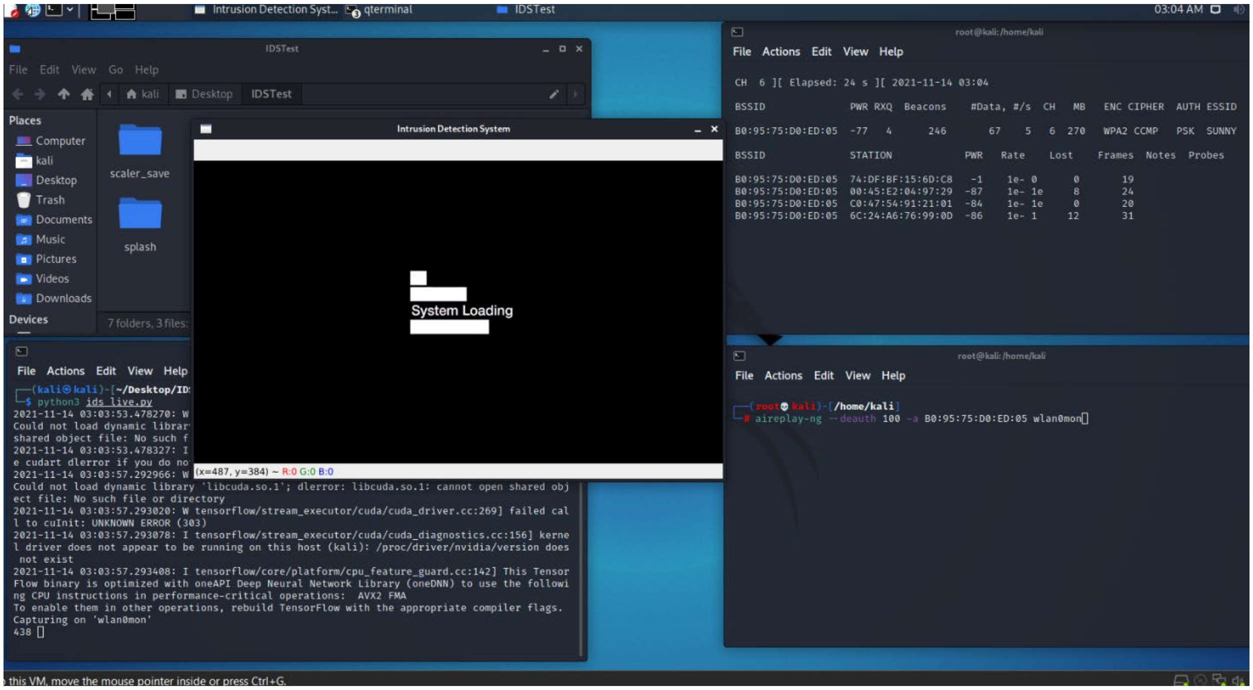
[Testing]



```
root@kali: /home/kali
┌──(root💀 kali)-[/home/kali]
└─# airodump-ng --bssid B0:95:75:D0:ED:05 -c 6 wlan0mon
```

```
kali@kali: ~/Desktop/IDSTest
┌──(kali㉿kali)-[~/Desktop/IDSTest]
└─$ python3 ids_live.py
```

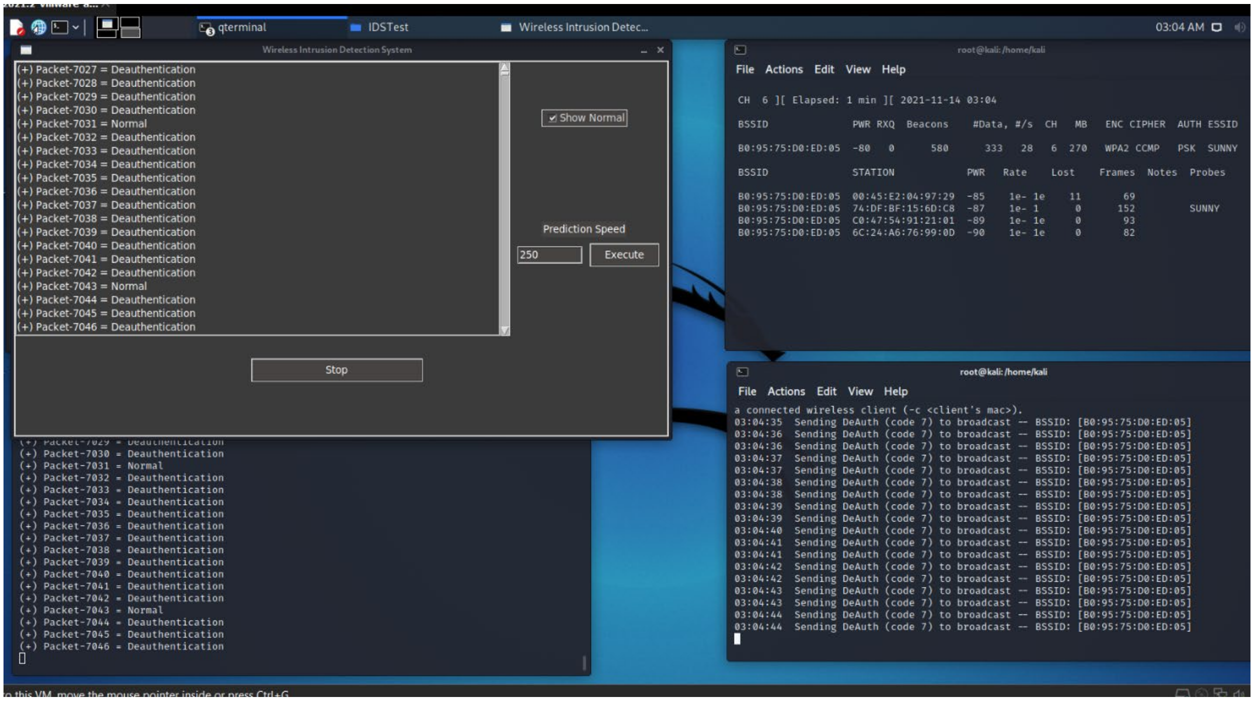
```
root@kali: /home/kali
┌──(root💀 kali)-[/home/kali]
└─# aireplay-ng --deauth 100 -a B0:95:75:D0:ED:05 wlan0mon
```



```
root@kali: /home/kali

 CH  6 ][ Elapsed: 24 s ][ 2021-11-14 03:04

 BSSID              PWR RXQ  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

 B0:95:75:D0:ED:05  -77   4      246       67    5   6  270   WPA2 CCMP   PSK  SUNNY

 BSSID              STATION            PWR   Rate    Lost    Frames  Notes  Probes

 B0:95:75:D0:ED:05  74:DF:BF:15:6D:C8   -1    1e- 0      0      19
 B0:95:75:D0:ED:05  00:45:E2:04:97:29  -87    1e- 1e     8      24
 B0:95:75:D0:ED:05  C0:47:54:91:21:01  -84    1e- 1e     0      20
 B0:95:75:D0:ED:05  6C:24:A6:76:99:0D  -86    1e- 1     12      31
```

```
┌──(kali㉿kali)-[~/Desktop/ID
└─$ python3 ids_live.py
2021-11-14 03:03:53.478270: W
Could not load dynamic librar
shared object file: No such f
2021-11-14 03:03:53.478327: I
e cudart dlerror if you do no
2021-11-14 03:03:57.292966: W
Could not load dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared obj
ect file: No such file or directory
2021-11-14 03:03:57.293020: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed cal
l to cuInit: UNKNOWN ERROR (303)
2021-11-14 03:03:57.293078: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kerne
l driver does not appear to be running on this host (kali): /proc/driver/nvidia/version does
 not exist
2021-11-14 03:03:57.293408: I tensorflow/core/platform/cpu_feature_guard.cc:142] This Tensor
Flow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the followi
ng CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
Capturing on 'wlan0mon'
438
```

```
root@kali: /home/kali
┌──(root💀 kali)-[/home/kali]
└─# aireplay-ng --deauth 100 -a B0:95:75:D0:ED:05 wlan0mon
```



qterminal
IDSTest
Wireless Intrusion Detec...
03:04 AM
Wireless Intrusion Detection System
☑ Show Normal
Prediction Speed
250
Execute
Stop
root@kali: /home/kali
File Actions Edit View Help

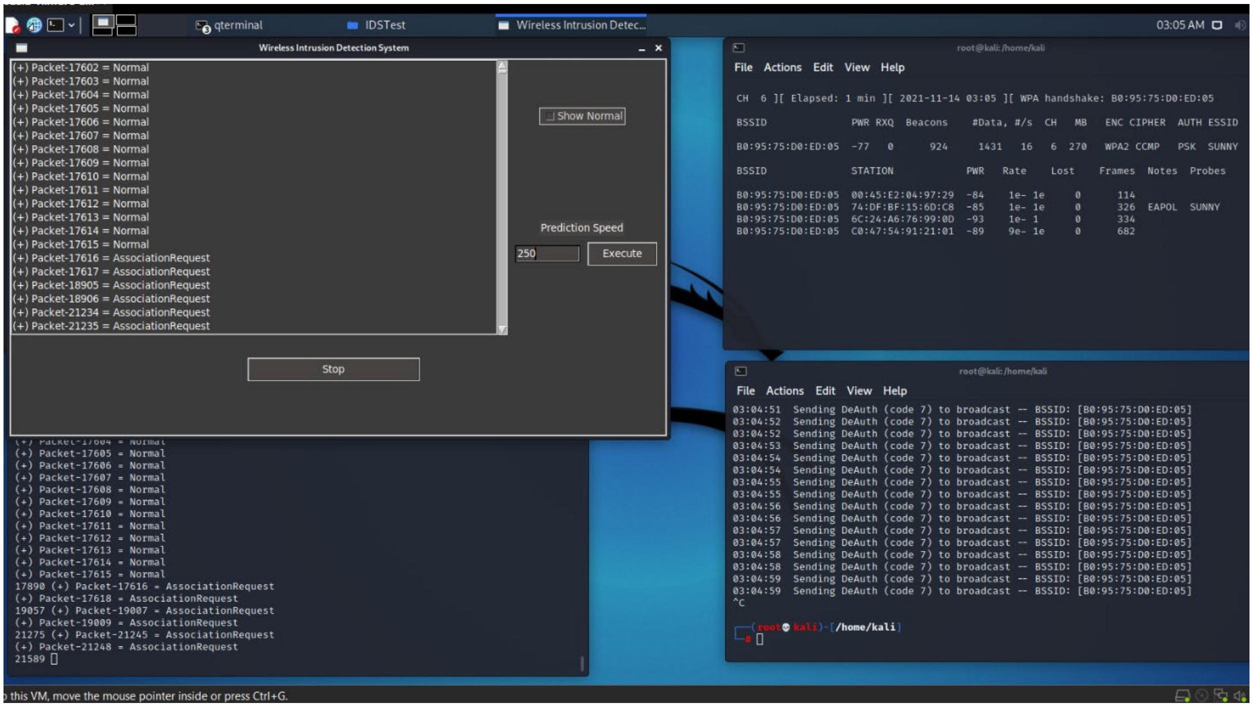
```
(+) Packet-7027 = Deauthentication
(+) Packet-7028 = Deauthentication
(+) Packet-7029 = Deauthentication
(+) Packet-7030 = Deauthentication
(+) Packet-7031 = Normal
(+) Packet-7032 = Deauthentication
(+) Packet-7033 = Deauthentication
(+) Packet-7034 = Deauthentication
(+) Packet-7035 = Deauthentication
(+) Packet-7036 = Deauthentication
(+) Packet-7037 = Deauthentication
(+) Packet-7038 = Deauthentication
(+) Packet-7039 = Deauthentication
(+) Packet-7040 = Deauthentication
(+) Packet-7041 = Deauthentication
(+) Packet-7042 = Deauthentication
(+) Packet-7043 = Normal
(+) Packet-7044 = Deauthentication
(+) Packet-7045 = Deauthentication
(+) Packet-7046 = Deauthentication
```

```
(+) Packet-7030 = Deauthentication
(+) Packet-7031 = Normal
(+) Packet-7032 = Deauthentication
(+) Packet-7033 = Deauthentication
(+) Packet-7034 = Deauthentication
(+) Packet-7035 = Deauthentication
(+) Packet-7036 = Deauthentication
(+) Packet-7037 = Deauthentication
(+) Packet-7038 = Deauthentication
(+) Packet-7039 = Deauthentication
(+) Packet-7040 = Deauthentication
(+) Packet-7041 = Deauthentication
(+) Packet-7042 = Deauthentication
(+) Packet-7043 = Normal
(+) Packet-7044 = Deauthentication
(+) Packet-7045 = Deauthentication
(+) Packet-7046 = Deauthentication
```

```
 CH  6 ][ Elapsed: 1 min ][ 2021-11-14 03:04

 BSSID              PWR RXQ  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

 B0:95:75:D0:ED:05  -80   0      580      333   28   6  270   WPA2 CCMP   PSK  SUNNY

 BSSID              STATION            PWR   Rate    Lost    Frames  Notes  Probes

 B0:95:75:D0:ED:05  00:45:E2:04:97:29  -85    1e- 1e    11       69
 B0:95:75:D0:ED:05  74:DF:BF:15:6D:C8  -87    1e- 1      0      152         SUNNY
 B0:95:75:D0:ED:05  C0:47:54:91:21:01  -89    1e- 1e     0       93
 B0:95:75:D0:ED:05  6C:24:A6:76:99:0D  -90    1e- 1e     0       82
```

```
a connected wireless client (-c <client's mac>).
03:04:35  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:36  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:36  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:37  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:37  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:38  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:38  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:39  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:39  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:40  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:41  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:41  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:42  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:42  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:43  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:43  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:44  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:44  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
```

o this VM, move the mouse pointer inside or press Ctrl+G.



qterminal
IDSTest
Wireless Intrusion Detec...
03:05 AM
Wireless Intrusion Detection System
□ Show Normal
Prediction Speed
250
Execute
Stop
root@kali: /home/kali
File Actions Edit View Help

```
(+) Packet-17602 = Normal
(+) Packet-17603 = Normal
(+) Packet-17604 = Normal
(+) Packet-17605 = Normal
(+) Packet-17606 = Normal
(+) Packet-17607 = Normal
(+) Packet-17608 = Normal
(+) Packet-17609 = Normal
(+) Packet-17610 = Normal
(+) Packet-17611 = Normal
(+) Packet-17612 = Normal
(+) Packet-17613 = Normal
(+) Packet-17614 = Normal
(+) Packet-17615 = Normal
(+) Packet-17616 = AssociationRequest
(+) Packet-17617 = AssociationRequest
(+) Packet-18905 = AssociationRequest
(+) Packet-18906 = AssociationRequest
(+) Packet-21234 = AssociationRequest
(+) Packet-21235 = AssociationRequest
```

```
(+) Packet-17605 = Normal
(+) Packet-17606 = Normal
(+) Packet-17607 = Normal
(+) Packet-17608 = Normal
(+) Packet-17609 = Normal
(+) Packet-17610 = Normal
(+) Packet-17611 = Normal
(+) Packet-17612 = Normal
(+) Packet-17613 = Normal
(+) Packet-17614 = Normal
(+) Packet-17615 = Normal
17890 (+) Packet-17616 = AssociationRequest
(+) Packet-17618 = AssociationRequest
19057 (+) Packet-19007 = AssociationRequest
(+) Packet-19009 = AssociationRequest
21275 (+) Packet-21245 = AssociationRequest
(+) Packet-21248 = AssociationRequest
21589 ▯
```

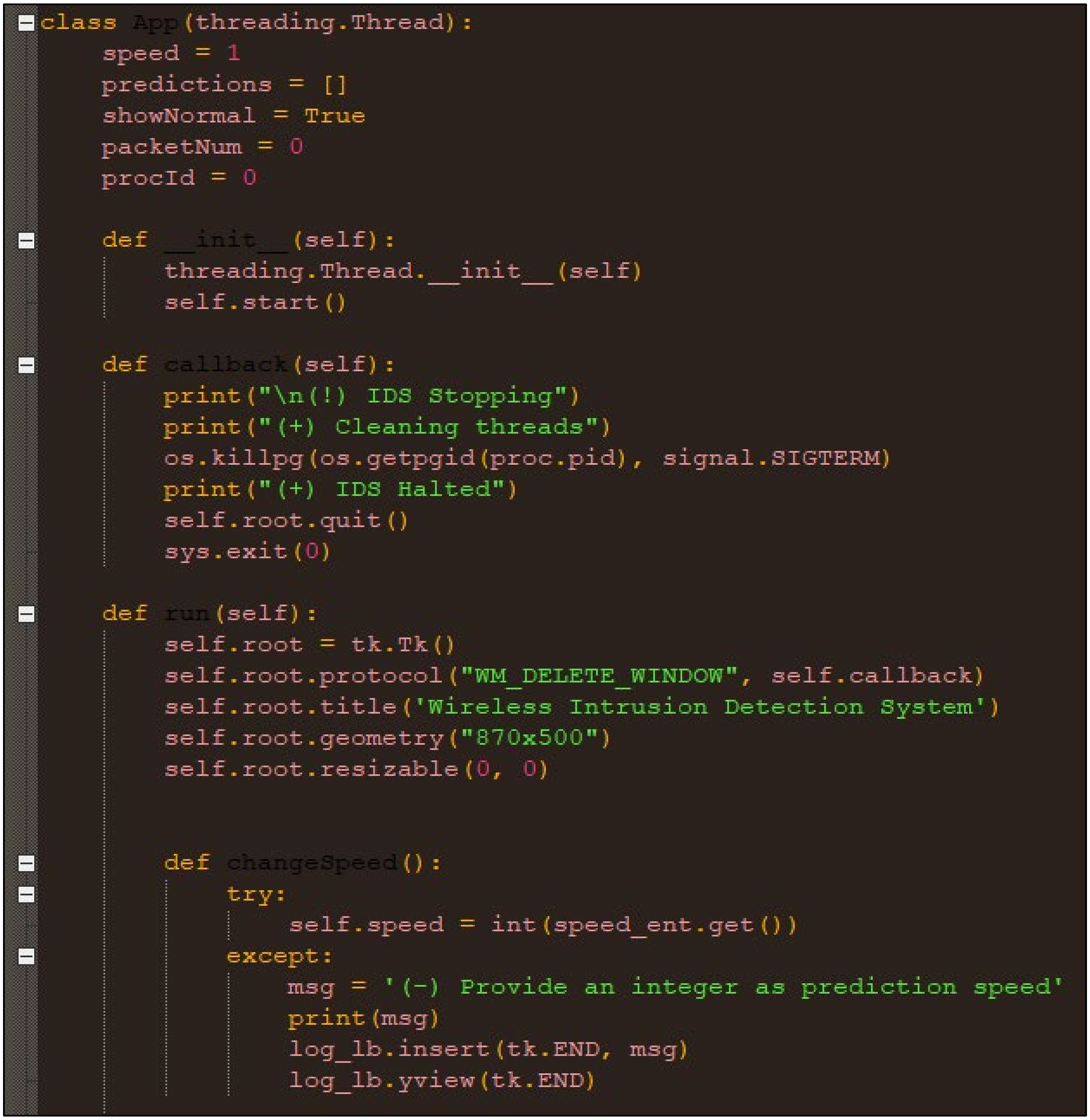
```
 CH  6 ][ Elapsed: 1 min ][ 2021-11-14 03:05 ][ WPA handshake: B0:95:75:D0:ED:05

 BSSID              PWR RXQ  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

 B0:95:75:D0:ED:05  -77   0      924     1431   16   6  270   WPA2 CCMP   PSK  SUNNY

 BSSID              STATION            PWR   Rate    Lost    Frames  Notes  Probes

 B0:95:75:D0:ED:05  00:45:E2:04:97:29  -84    1e- 1e     0      114
 B0:95:75:D0:ED:05  74:DF:BF:15:6D:C8  -85    1e- 1e     0      326  EAPOL  SUNNY
 B0:95:75:D0:ED:05  6C:24:A6:76:99:0D  -93    1e- 1      0      334
 B0:95:75:D0:ED:05  C0:47:54:91:21:01  -89    9e- 1e     0      682
```

```
03:04:51  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:52  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:52  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:53  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:54  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:54  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:55  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:55  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:56  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:56  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:57  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:57  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:58  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:58  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:59  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
03:04:59  Sending DeAuth (code 7) to broadcast -- BSSID: [B0:95:75:D0:ED:05]
^C

┌──(root💀kali)-[/home/kali]
└─# ▯
```

o this VM, move the mouse pointer inside or press Ctrl+G.

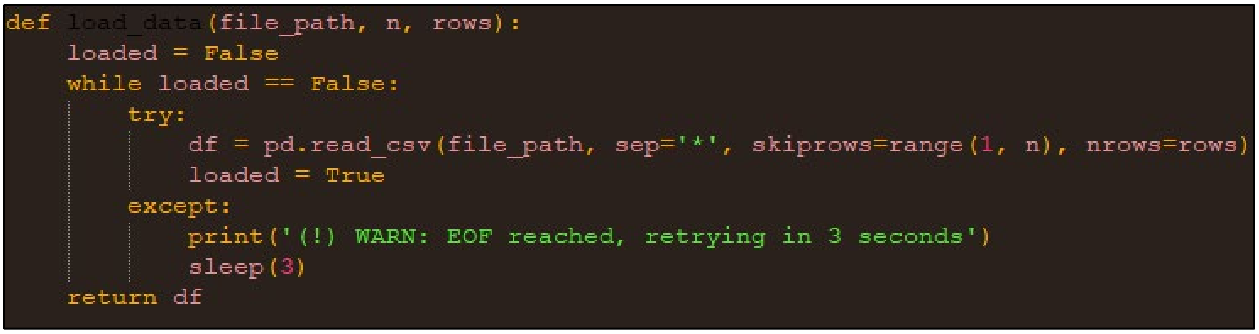
An interesting thing to note here is that after stopping deauthentication attack, WPA Handshake has been captured as visible in the top left terminal and in the IDS as 4-5 Association Request frames. This can be avoided using such an IDS by sending a fake WPA Handshake frame, and simultaneously deauthenticating our own clients for a little more time just after the attacker stops the burst deauthentication attack.

5. Code Snippets



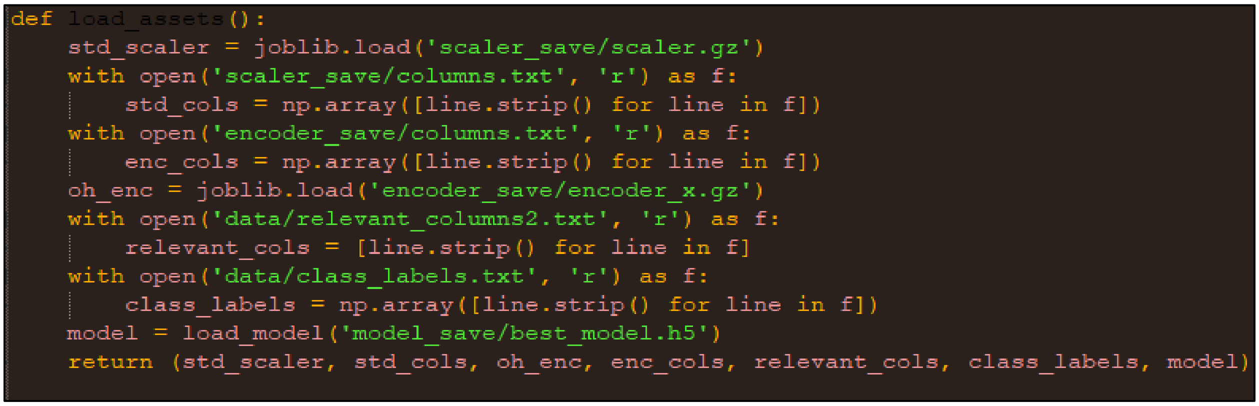
```
class App(threading.Thread):
    speed = 1
    predictions = []
    showNormal = True
    packetNum = 0
    procId = 0

    def __init__(self):
        threading.Thread.__init__(self)
        self.start()

    def callback(self):
        print("\n(!) IDS Stopping")
        print("(+) Cleaning threads")
        os.killpg(os.getpgid(proc.pid), signal.SIGTERM)
        print("(+) IDS Halted")
        self.root.quit()
        sys.exit(0)

    def run(self):
        self.root = tk.Tk()
        self.root.protocol("WM_DELETE_WINDOW", self.callback)
        self.root.title('Wireless Intrusion Detection System')
        self.root.geometry("870x500")
        self.root.resizable(0, 0)


        def changeSpeed():
            try:
                self.speed = int(speed_ent.get())
            except:
                msg = '(-) Provide an integer as prediction speed'
                print(msg)
                log_lb.insert(tk.END, msg)
                log_lb.yview(tk.END)
```

This is the class for handling GUI, interactive operations and detect change of state, which runs on separate thread. Here we also see a callback for safe exiting which sends interrupts to all threads if stop button or window is closed by user. Further down the class, a simple, scalable layout is defined (omitted from the snip)



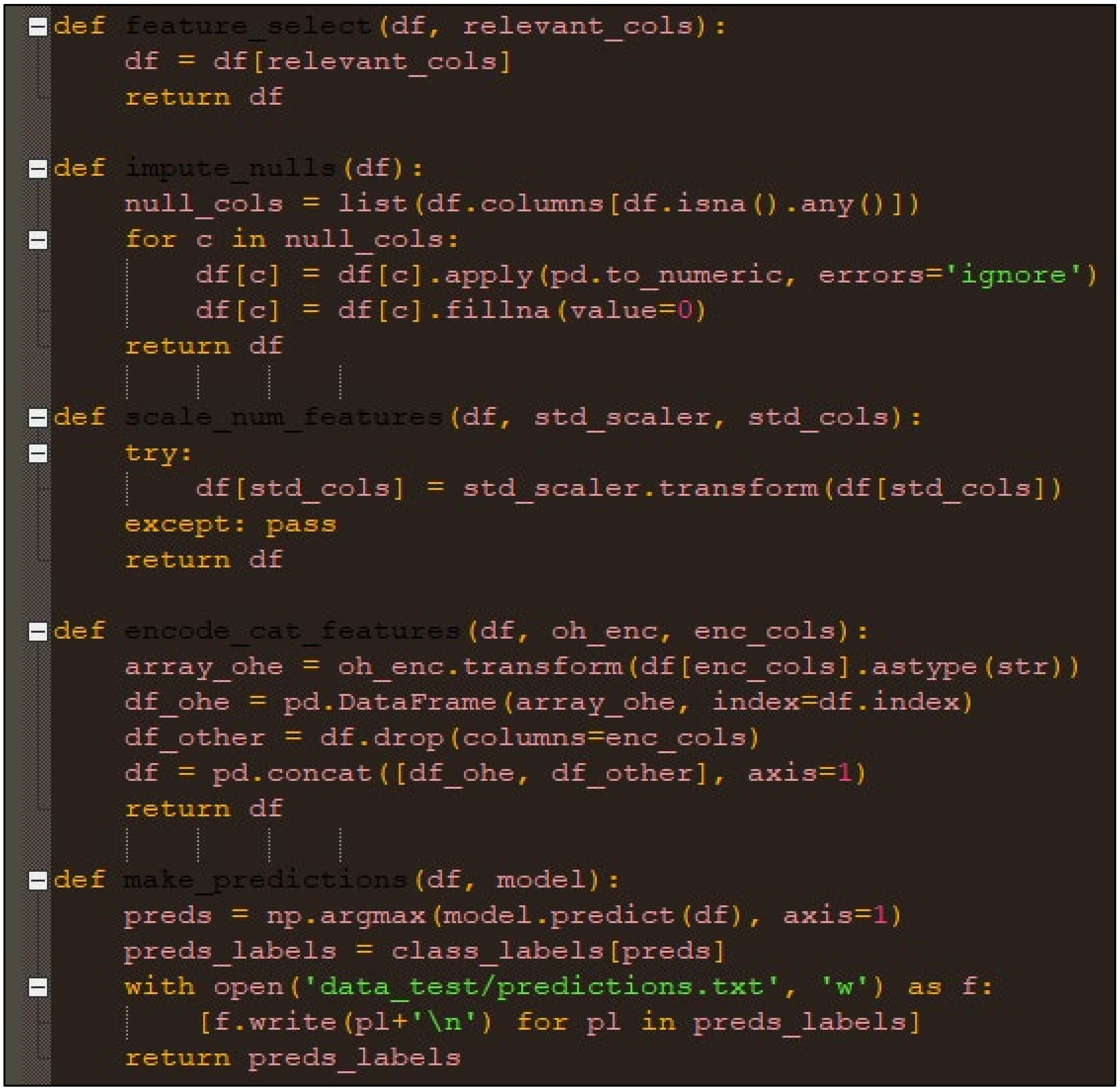
```python
def load_data(file_path, n, rows):
    loaded = False
    while loaded == False:
        try:
            df = pd.read_csv(file_path, sep='*', skiprows=range(1, n), nrows=rows)
            loaded = True
        except:
            print('(!) WARN: EOF reached, retrying in 3 seconds')
            sleep(3)
    return df
```

This function constantly loads data by nrows parameter (used as speed) from the CSV file which is constantly being updated by tshark running as sub process on different thread



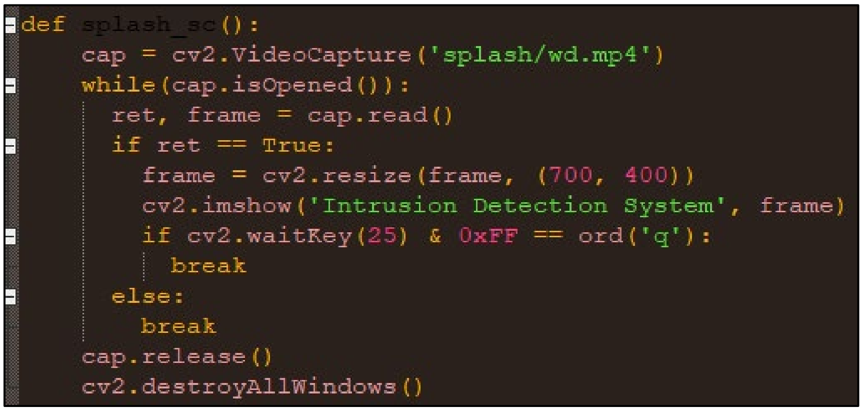
```python
def load_assets():
    std_scaler = joblib.load('scaler_save/scaler.gz')
    with open('scaler_save/columns.txt', 'r') as f:
        std_cols = np.array([line.strip() for line in f])
    with open('encoder_save/columns.txt', 'r') as f:
        enc_cols = np.array([line.strip() for line in f])
    oh_enc = joblib.load('encoder_save/encoder_x.gz')
    with open('data/relevant_columns2.txt', 'r') as f:
        relevant_cols = [line.strip() for line in f]
    with open('data/class_labels.txt', 'r') as f:
        class_labels = np.array([line.strip() for line in f])
    model = load_model('model_save/best_model.h5')
    return (std_scaler, std_cols, oh_enc, enc_cols, relevant_cols, class_labels, model)
```

This function loads all the assets we trained and fitted on the dataset before like scalar, encoder, columns, labels and the model itself.



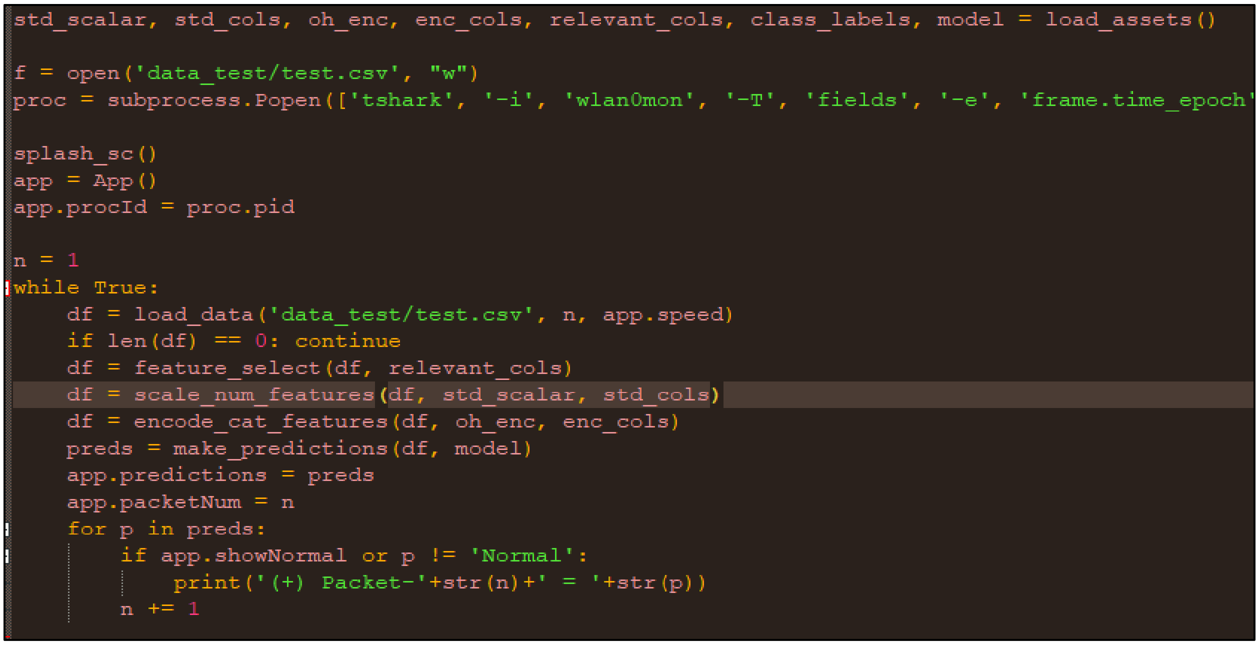
```python
def feature_select(df, relevant_cols):
    df = df[relevant_cols]
    return df

def impute_nulls(df):
    null_cols = list(df.columns[df.isna().any()])
    for c in null_cols:
        df[c] = df[c].apply(pd.to_numeric, errors='ignore')
        df[c] = df[c].fillna(value=0)
    return df

def scale_num_features(df, std_scaler, std_cols):
    try:
        df[std_cols] = std_scaler.transform(df[std_cols])
    except: pass
    return df

def encode_cat_features(df, oh_enc, enc_cols):
    array_ohe = oh_enc.transform(df[enc_cols].astype(str))
    df_ohe = pd.DataFrame(array_ohe, index=df.index)
    df_other = df.drop(columns=enc_cols)
    df = pd.concat([df_ohe, df_other], axis=1)
    return df

def make_predictions(df, model):
    preds = np.argmax(model.predict(df), axis=1)
    preds_labels = class_labels[preds]
    with open('data_test/predictions.txt', 'w') as f:
        [f.write(pl+'\n') for pl in preds_labels]
    return preds_labels
```

After loading the data, all these functions run in a sequence using the assets loaded before, and predictions are made for that data.



```python
def splash_sc():
    cap = cv2.VideoCapture('splash/wd.mp4')
    while(cap.isOpened()):
      ret, frame = cap.read()
      if ret == True:
        frame = cv2.resize(frame, (700, 400))
        cv2.imshow('Intrusion Detection System', frame)
        if cv2.waitKey(25) & 0xFF == ord('q'):
          break
      else:
        break
    cap.release()
    cv2.destroyAllWindows()
```

This function shows a loading video till some packets load into the CSV buffer file



```python
std_scalar, std_cols, oh_enc, enc_cols, relevant_cols, class_labels, model = load_assets()

f = open('data_test/test.csv', "w")
proc = subprocess.Popen(['tshark', '-i', 'wlan0mon', '-T', 'fields', '-e', 'frame.time_epoch'

splash_sc()
app = App()
app.procId = proc.pid

n = 1
while True:
    df = load_data('data_test/test.csv', n, app.speed)
    if len(df) == 0: continue
    df = feature_select(df, relevant_cols)
    df = scale_num_features(df, std_scalar, std_cols)
    df = encode_cat_features(df, oh_enc, enc_cols)
    preds = make_predictions(df, model)
    app.predictions = preds
    app.packetNum = n
    for p in preds:
        if app.showNormal or p != 'Normal':
            print('(+) Packet-'+str(n)+' = '+str(p))
        n += 1
```

This is the ‘main’ of the program, where all the functions discussed are executed.

First asset resources are loaded, a test.csv file is opened, then tshark command is executed as a sub-process which runs on the monitor node and extracts relevant features from the packets and writes these into the CSV file.

Loading splash screen is displayed, then class is initialized, processId is set to tshark process so that, the tshark process can be terminated from class callback functions gracefully.

Finally in a never ending loop, data is loaded from the CSV, cleaned, scaled, encoded, predicted and printed to GUI instantaneously.

6. GitHub Link

https://github.com/BinitDOX/Wireless-Intrusion-Detection-System