imports

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os
import re
from pathlib import Path
```

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [3]:
```
Path.cwd()
```

Out[3]:
```
PosixPath('/content')
```

In [4]:
```
import sys
sys.path.append("/content/drive/MyDrive/colab_notebooks")
```

In [5]:
```
import utils as ut
```

In [46]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Dropout, Flatten, Conv1D, Max
import tensorflow as tf
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from keras.models import load_model
```

Data Set

https://www.stratosphereips.org/datasets-iot23

Reading in the Data and reformatting it

Here we are reading in all the raw packet data that has been exported out into multiple csv files.

In [7]:
```
data_folder = Path(r'/content/drive/MyDrive/usd classes/intro to ai/Pcap Data Proce
```

In [8]:
```
hex_dump_files = []
label_files = []
```

```
file_dic = {}

for path in data_folder.iterdir():
  file_name = path.name

  if file_name.endswith('Hexdump.csv'):

    file_tag = "_".join(file_name.split('_')[:2])

    if file_tag not in file_dic:
      file_dic[file_tag] = {}

    file_dic[file_tag]['hex_file'] = path

    #hex_dump_files.append(path)

  if file_name.endswith('.labeled'):
    file_tag = file_name.split('.')[0]
    if file_tag not in file_dic:
      file_dic[file_tag] = {}

    file_dic[file_tag]['label_file'] = path

    #label_files.append(path)
```

In [ ]:
```
#merge_df.head()
```

In [9]:
```
hex_conversion = {
    "0":0,
    "1":1,
    "2":2,
    "3":3,
    "4":4,
    "5":5,
    "6":6,
    "7":7,
    "8":8,
    "9":9,
    "A":10,
    "B":11,
    "C":12,
    "D":13,
    "E":14,
    "F":15
}
```

In [10]:
```
def convert_hex_to_dec(hex_num:str)->int:
    byte_d = 0

    for i, b in enumerate(hex_num, 1):
        decimal_position = len(hex_num) - i
        b_to_d = hex_conversion[b]
        value = b_to_d * 16**decimal_position
        byte_d +=value
```

```
    return int(byte_d)
```

In [11]:
```
def get_byte_list(byte_str)->list:
    real_byte = []

    for byte in byte_str.split(' '):
        match = re.match(r"^[A-Fa-f0-9]+$", byte)

        if (len(byte) == 2) and match:

            decimal_byte = convert_hex_to_dec(byte.upper())

            real_byte.append(decimal_byte)

    return real_byte
```

In [12]:
```
def setup_raw_bytes_df(df:pd.DataFrame)->pd.DataFrame:
    df_index_ts = []
    list_of_lists = []


    for ts, hexdump in zip(df.ts, df.hexdump):

        df_index_ts.append(ts)
        list_of_bytes = get_byte_list(hexdump)

        list_of_lists.append(list_of_bytes)

    raw_df = pd.DataFrame(index = df_index_ts, data = list_of_lists)
    #raw_df = raw_df.astype(int)
    raw_df['label'] = df.label.tolist()
    raw_df = raw_df.reset_index().rename(columns = {'index':'ts'})

    return raw_df
```

In [13]:
```
def combine_raw_and_labeled(
    label_file_path: Path,
    raw_file_path:Path
  )-> pd.DataFrame:

  label_df = ut.read_and_parse_label_file(label_file_path)
  raw_packets = pd.read_csv(raw_file_path)

  raw_packets.rename(columns = {'epoch_time': 'ts'}, inplace = True)
  label_df['ts'] = label_df.ts.astype(float)

  merge_df = raw_packets.merge(label_df,
                 on = 'ts',
                 how = 'outer',
                 indicator = True
                )

  complete_data_df = merge_df[merge_df._merge == 'both'].copy()
```

```
  raw_df = setup_raw_bytes_df(complete_data_df)

  return raw_df
```

Iterating through and reading in the first three captrure files.

In [14]:
```
all_combined_files = []

for i, label_name in enumerate(file_dic):
  if i < 3:
    print(label_name)
    one_dic = file_dic[label_name]
    if 'label_file' and 'hex_file' in one_dic:
      print('processing ', label_name)
      label_file = one_dic['label_file']
      hex_file = one_dic['hex_file']

      comb_df = combine_raw_and_labeled(label_file_path=label_file,
                                        raw_file_path=hex_file
                                       )

      comb_df['file_label'] = label_name
      #comb_df.to_csv(f'raw_data/{label_name}_combined_data.csv', index = False)

      all_combined_files.append(comb_df)
      print('')
```

```
Malware_3
processing  Malware_3
/content/drive/MyDrive/colab_notebooks/utils.py:5: DtypeWarning: Columns (0,3,5,14,16,17,18,19,21) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv(path_to_label_file,
Malware_1
processing  Malware_1
/content/drive/MyDrive/colab_notebooks/utils.py:5: DtypeWarning: Columns (0,3,5,14,16,17,18,19,21) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv(path_to_label_file,
Malware_8
processing  Malware_8
```

In [15]:
```
all_data_df = pd.concat(all_combined_files)
```

In [16]:
```
save_csv_path = r'/content/drive/MyDrive/usd classes/intro to ai/truncated_1_170_00
```

In [17]:
```
all_data_df.to_csv('/content/drive/MyDrive/usd classes/intro to ai/truncated_1_170_
```

In [18]:
```
all_data_df['label'] = all_data_df.label.replace({' Benign':0,
                          ' Malicious':1
                         })

all_data_df.fillna(int(0), inplace=True) # any NA values will be filled with zeros
```

```
<ipython-input-18-cd206d75aefb>:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  all_data_df['label'] = all_data_df.label.replace({' Benign':0,
```

Setting aside validation data. That way we can pass a new capture file to the model that was not in the original training data.

In [23]:
```
validation_df = all_data_df[all_data_df.file_label == all_data_df.file_label.unique
non_validation_df = all_data_df[all_data_df.file_label != all_data_df.file_label.un
```

Setting up the data to be passed into the CNN

In [24]:
```
len(validation_df)
```

Out[24]: 10403

In [25]:
```
X = non_validation_df.drop(columns=['ts', 'label', 'file_label']).copy()
y = non_validation_df.label
```

Data Exploration

In [26]:
```
non_validation_df.label.value_counts() / len(non_validation_df)
```

Out[26]:

label	count
1	0.593243
0	0.406757

dtype: float64

Setting up the input and output data (X and y). Also scaling the the input data for both the training and validation data

In [27]:
```
X_val_add = validation_df.drop(columns=['ts', 'label', 'file_label']).copy()
y_val_add = validation_df.label
```

In [28]:
```
scaler = MinMaxScaler()
scaler.fit(X_val_add)
X_val_add_scaled = scaler.transform(X_val_add)
```

In [29]:
```
X_val_add_scaled
```

Out[29]:
```
array([[0., 1., 0., ..., 0., 0., 0.],
       [1., 0., 1., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       ...,
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.]])
```

In [30]:
```
X.head()
```

Out[30]:

	0	1	2	3	4	5	6	7	8	9	...	357	358	359	360	361	362
0	166	209	140	31	206	100	184	39	235	199	...	0.0	0.0	0.0	0.0	0.0	0.0
1	166	209	140	31	206	100	184	39	235	199	...	0.0	0.0	0.0	0.0	0.0	0.0
2	166	209	140	31	206	100	184	39	235	199	...	0.0	0.0	0.0	0.0	0.0	0.0
3	184	39	235	199	233	201	166	209	140	31	...	0.0	0.0	0.0	0.0	0.0	0.0
4	166	209	140	31	206	100	184	39	235	199	...	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 367 columns

In [31]:
```
X.shape
```

Out[31]:
```
(1164852, 367)
```

In [32]:
```
scaler = MinMaxScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

In [32]:
```

```

In [33]:
```
# splitting our data into test and training
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
                                                    test_size=0.2,
                                                    random_state=37
                                                   )
```

In [34]:
```
counts_df = pd.DataFrame(y_train.value_counts())
```

In [35]:
```
counts_df['proportion'] = round(counts_df['count'] / len(y_train), 3)
```

In [36]:
```
counts_df.head()
```

Out[36]:

label	count	proportion
1	552772	0.593
0	379109	0.407

In [37]:
```
len(X_test)
```

Out[37]: 232971

setting up the CNN model

In [38]:
```
input_shape = X_train.shape[1]
```

In [39]:
```
input_shape
```

Out[39]: 367

In [40]:
```
import torch
assert torch.cuda.is_available(), "GPU not available"
```

In [41]:
```
model = Sequential()
kernel = (3)
model.add(Conv1D(32, kernel_size = kernel, activation = 'relu', input_shape=(input_
model.add(Conv1D(64, kernel_size = kernel, activation = 'relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.25))

model.add(Conv1D(64, kernel_size = kernel, activation = 'relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.25))

model.add(Conv1D(128, kernel_size = kernel, activation = 'relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.25))

model.add(Flatten()) # is this redundant lol
model.add(Dense(64, activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:
107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [42]:
```
adam = keras.optimizers.Adam(learning_rate=0.001)
```

In [43]:
```
model.compile(loss='binary_crossentropy', optimizer=adam, metrics=["accuracy"])
```

training the model

In [ ]:
```
model_history = model.fit(X_train,
                          y_train,
                          batch_size=100,
                          validation_split=.10,
                          epochs = 50
                          )
```

In [44]:
```
model_path = "/content/drive/MyDrive/usd classes/intro to ai/cnn_model_1_170_000_sa
```

In [ ]:
```
model.save(model_path)
```

In [47]:
```
model_loaded = load_model(model_path)
```

In [ ]:
```
model_history.history.keys()
```

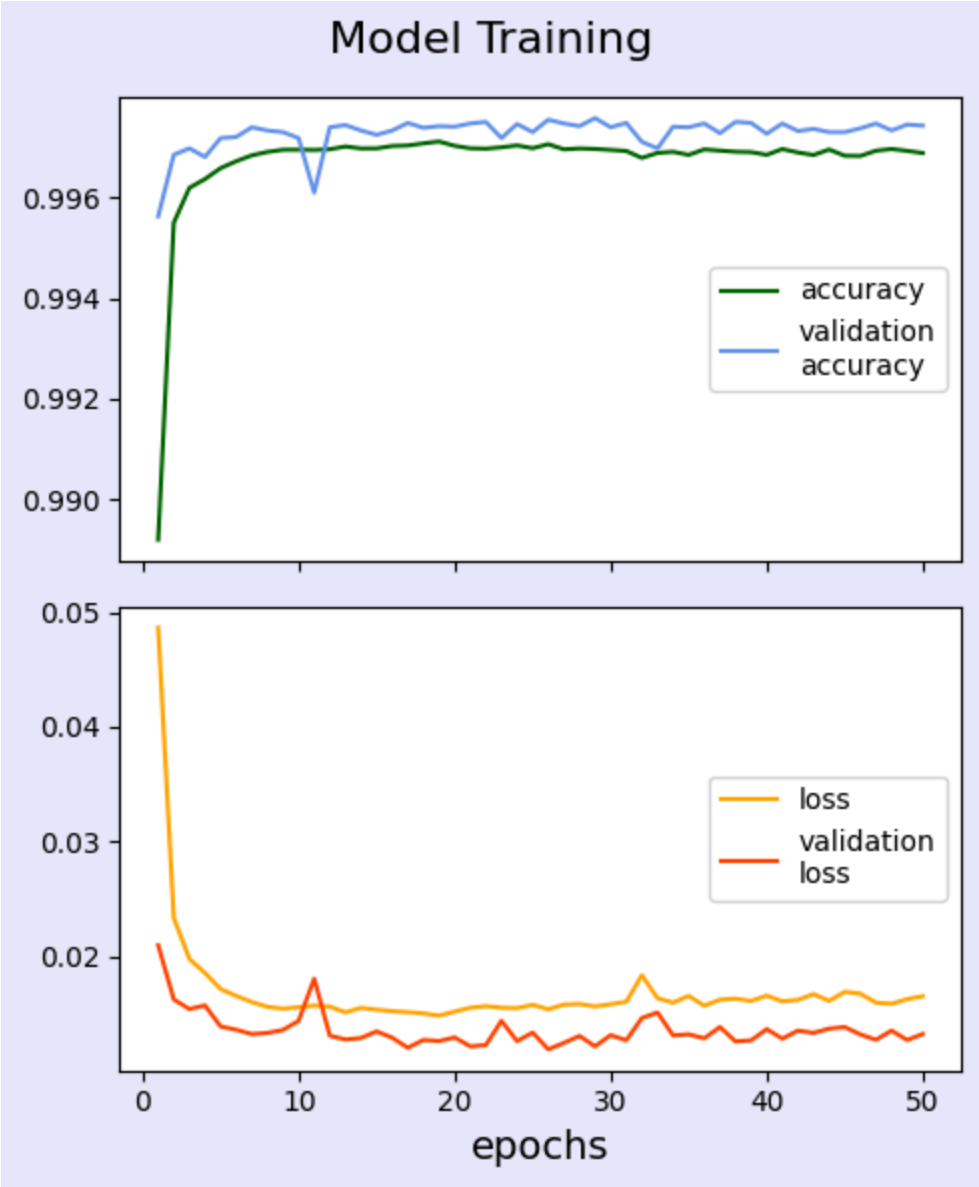
Out[ ]:
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

In [ ]:
```
x_axis = np.arange(1, 51, 1) # 50 epochs
```

In [ ]:
```
fig, ax = plt.subplots(2, figsize = (5,6), sharex=True)

ax[0].plot(x_axis, model_history.history['accuracy'],
         color = 'darkgreen',
         label='accuracy'
         )
ax[0].plot(x_axis, model_history.history['val_accuracy'],
         color = 'cornflowerblue',
         label='validation\naccuracy'
         )
ax[1].plot(x_axis, model_history.history['loss'],
         color = 'orange',
         label='loss'
         )
ax[1].plot(x_axis, model_history.history['val_loss'],
         color = 'orangered',
         label='validation\nloss'
         )
ax[1].set_xlabel('epochs', fontsize=14)
ax[0].legend(loc='center right')
ax[1].legend(loc='center right')

ax[0].set_aspect('auto')
ax[1].set_aspect('auto')
fig.suptitle('Model Training', fontsize=16)
fig.patch.set_facecolor('lavender')
plt.tight_layout()
plt.savefig('cnn training.jpeg', dpi=600, bbox_inches='tight')
```



evaluating the model

In [ ]:
```
loss_and_metrics = model_loaded.evaluate(X_test, y_test)
```

```
7281/7281 ━━━━━━━━━━━━━━━━━━━━ 14s 2ms/step - accuracy: 0.9972 - loss: 0.0147
```

In [ ]:
```
print(f'Loss = {loss_and_metrics[0]}')
print(f'Accuracy = {loss_and_metrics[1]}')
```

```
Loss = 0.014361095614731312
Accuracy = 0.9972872138023376
```

In [ ]:
```
X_test.shape
```

Out[ ]:
```
(232971, 367)
```

In [ ]:
```
y_test.shape
```

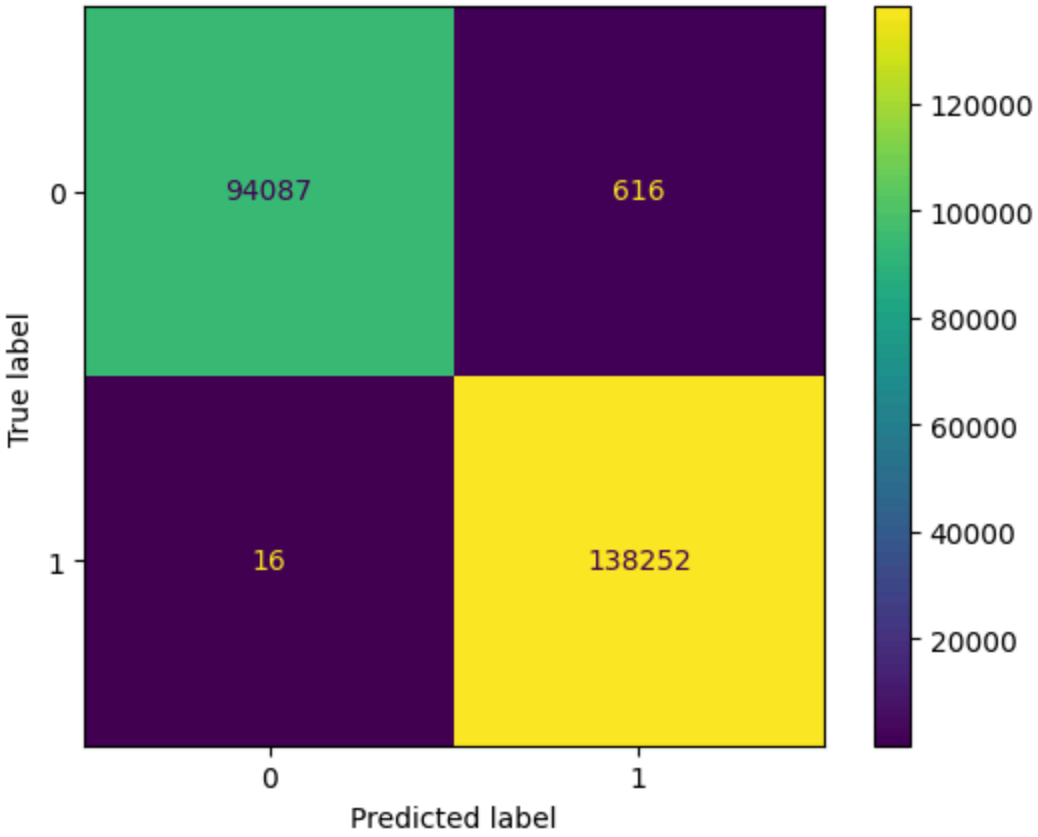
Out[ ]:
```
(232971,)
```

In [ ]:
```
predicted = model_loaded.predict(X_test)
```

```
7281/7281 ━━━━━━━━━━━━━━━━━━━━ 11s 1ms/step
```

In [ ]:
```
predicted = tf.squeeze(predicted)
predicted = np.array([1 if x >= 0.5 else 0 for x in predicted])
actual = np.array(y_test)
conf_mat = confusion_matrix(actual, predicted)
displ = ConfusionMatrixDisplay(confusion_matrix=conf_mat)
displ.plot()
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b6ae8c4fa10>
```



New Capture File

In [ ]:
```
X_val_add_scaled.shape
```

Out[ ]:
```
(10403, 367)
```

In [ ]:
```
y_val_add.shape
```

Out[ ]:
```
(10403,)
```

In [ ]:
```
loss_and_metrics = model_loaded.evaluate(X_val_add_scaled, y_val_add)
print(f'Loss = {loss_and_metrics[0]}')
print(f'Accuracy = {loss_and_metrics[1]}')
```

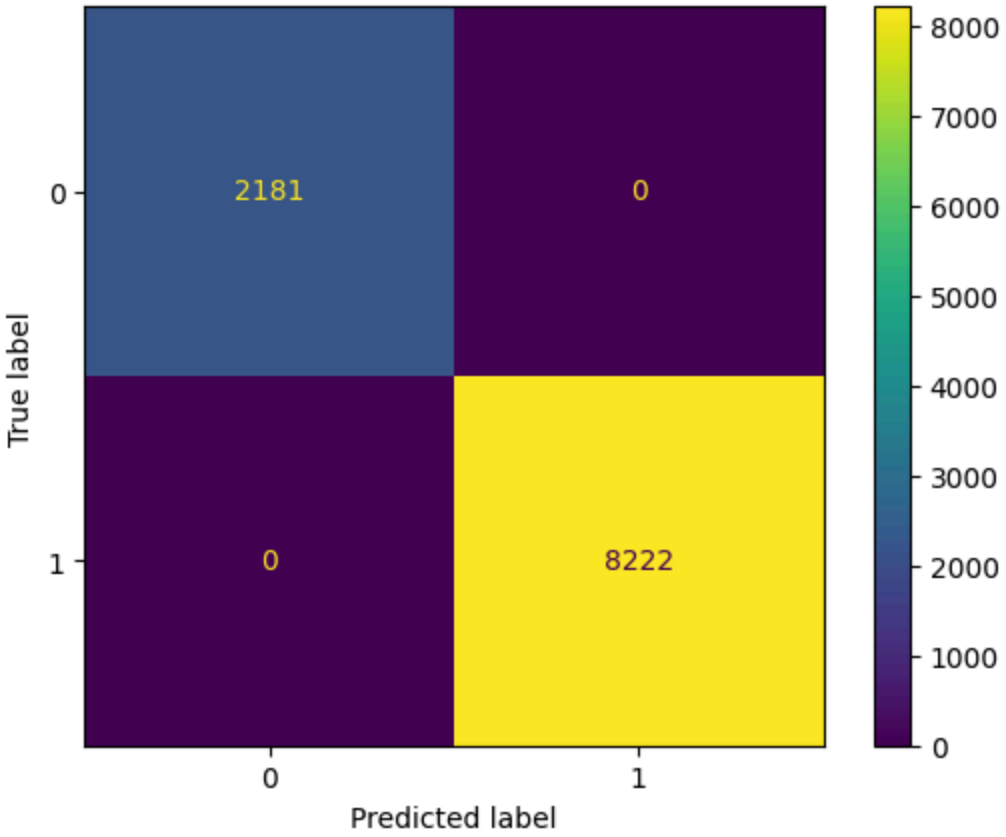
```
326/326 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 1.0000 - loss: 0.0316
Loss = 0.034200023859739304
Accuracy = 1.0
```

In [ ]:
```
predicaiton_validation = model_loaded.predict(X_val_add_scaled)
```

```
326/326 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```

In [ ]:
```
predicted = tf.squeeze(predicaiton_validation)
predicted = np.array([1 if x >= 0.5 else 0 for x in predicted])
actual = np.array(y_val_add)
conf_mat = confusion_matrix(actual, predicted)
displ = ConfusionMatrixDisplay(confusion_matrix=conf_mat)
displ.plot()
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b6ae8fb8990>



Another validation on a new file

In [ ]:
```
label_file = file_dic['Malware_20']['label_file']
hex_file = file_dic['Malware_20']['hex_file']

comb_df = combine_raw_and_labeled(label_file_path=label_file,
                                  raw_file_path=hex_file
                                 )
```

In [ ]:
```
comb_df = pd.concat([comb_df])
```

In [ ]:
```
comb_df['label'] = comb_df.label.replace({' Benign':0,
                          ' Malicious':1
```

```
                        })

comb_df.fillna(int(0), inplace=True)
```

```
<ipython-input-169-48bcdf520963>:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  comb_df['label'] = comb_df.label.replace({' Benign':0,
```

In [ ]:
```
comb_df.label.unique()
```

Out[ ]:
```
array([0, 1])
```

In [ ]:
```
comb_df.shape
```

Out[ ]:
```
(3209, 104)
```

In [ ]:
```
zero_df = pd.DataFrame(np.zeros((comb_df.shape[0], 265)))
```

In [ ]:
```
zero_df
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	...	255	256	257	258	259	260	26
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
3204	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
3205	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
3206	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
3207	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
3208	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.

3209 rows × 265 columns

In [ ]:
```
X = comb_df.drop(columns=['ts', 'label']).copy()
y = comb_df.label
```

In [ ]:
```
X = X.astype(int)
```

In [ ]:
```
X.shape
```

Out[ ]: (3209, 102)

In [ ]:
```
X = X.merge(zero_df, how = 'outer', left_index = True, right_index = True,)
X.columns = X.columns.astype(str)
```

In [ ]:
```
scaler = MinMaxScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

In [ ]:
```
y.dtypes
```

Out[ ]: dtype('int64')

In [ ]:
```
X.shape
```

Out[ ]: (3209, 367)

In [ ]:
```
loss_and_metrics = model_loaded.evaluate(X_scaled, y)
print(f'Loss = {loss_and_metrics[0]}')
print(f'Accuracy = {loss_and_metrics[1]}')
```

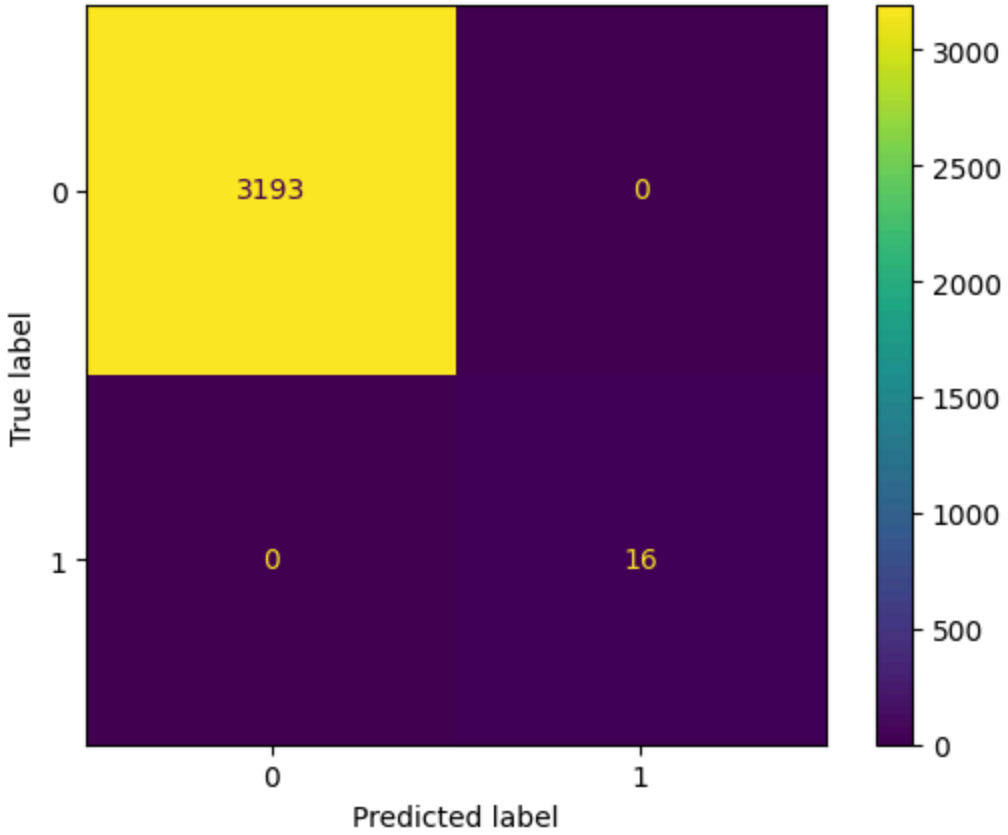
```
101/101 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 1.0000 - loss: 0.0021
Loss = 0.0005326070822775364
Accuracy = 1.0
```

In [ ]:
```
predicaiton_validation = model_loaded.predict(X_scaled)
```

```
101/101 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
```

In [ ]:
```
predicted = tf.squeeze(predicaiton_validation)
predicted = np.array([1 if x >= 0.5 else 0 for x in predicted])
actual = np.array(y)
conf_mat = confusion_matrix(actual, predicted)
displ = ConfusionMatrixDisplay(confusion_matrix=conf_mat)
displ.plot()
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b658a4519d0>


3193
0
0
16
0
1
True label
0
1
Predicted label
3000
2500
2000
1500
1000
500
0

In [ ]:

In [ ]: