WEB APPLICATION FIREWALL USING MACHINE LEARNING

A PROJECT REPORT

Submitted in partial fulfillment of the requirement for the award of the Degree of

BACHELOR OF TECHNOLOGY
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

by

Utkarsh Maurya (19BEC1308)

Under the Guidance of

Dr. Vijaykumar P

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF ELECTRONICS ENGINEERING
VELLORE INSTITUTE OF TECHNOLOGY
CHENNAI - 600127

May 2022

ABSTRACT

Internet application firewalls (WAFs) are usually used to hit upon (and now and again block) web assaults. Many industrial WAFs are to be had, in conjunction with several freely available (typically open supply) alternatives. WAFs can be difficult to personalize for a selected utility, making it hard to run them in “whitelisting mode.” it is commonplace to find WAFs deployed in “blacklisting mode,” making them greater at risk of bypasses and centered assaults. maximum open supply WAFs have a publicly reachable demo software showing the effectiveness of their filtering, and on occasion the WAF's administrative interface as well.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ACKNOWLEDGEMENT	**3**
	ABSTRACT	**4**
	LIST OF FIGURES	**6**
1	**INTRODUCTION**	**8**
	1.1 Preamble	**8**
	1.2 Objectives	
	1.3 Background and Literature Survey	**8**
	1.4 Need for Web Application Firewall	**9**
	1.5 Organization of the Report	**10**
2	**WEB APPLICATION FIREWALL**	**12**
	2.1 Methodology	**12**
	2.2 Design Approach	**15**
	2.3 Database Configuration Firewall	**17**
	2.4 Advantages	**18**
3	**RESULT AND DISCUSSIONS**	**19**
	3.1 Anomaly Detection	**19**
	3.2 Training an Intrusion Detection System	**22**
4	**CONCLUSION AND FUTURE WORK**	**24**
	4.1 Conclusion	**24**
	4.2 Future Work	**24**
	APPENDIX	**25**
	REFERENCES	**32**

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
1.1	Physical Block diagram	8
1.2	External Design Flowchart	9
1.3	Database Configuration of Firewall	10
2.1	Reading the Dataset	15
2.2	Indication of attack	16
2.3	Preprocessing of the data	16
2.4	Division of the data	17
2.5	Type of attack indicated	18
2.6	Clustering	19
2.7	Training the Autoencoder	20
2.9	Anomaly Detection	21
2.10	Analyzing the Dataset	22
2.11	Encoding the Feature Vector	22
2.12	Conversion of data into Z-score and dummy variable	23
2.13	Grouping of Dataset	23
2.14	Training of Neural Network	24
2.15	Accuracy Measurement	24

LIST OF ABBREVIATION

ABBREVIATION	TITLE	PAGE NO.
WAF	Web Application Firewall	4
SQL	Structured Query Language	8
XSS	Cross Site Scripting	8
DDoS	Distributed Denial of Service	8
HIPAA	Health Insurance Portability and Accountability Act	8
PCI	Pharmacy Council of India	8
TSNE	T-distributed Stochastic Neighbourhood Embedding	20
RSME	Root means square error	22

LIST OF TABLES

TABLE NO.	TITLE	PAGE NO.
1.1	Literature Survey	10

CHAPTER 1

INTRODUCTION

1.1 PREAMBLE

Web applications commonly have security flaws and vulnerabilities that, if exploited, expose the user's data and compromise the application. Vulnerabilities such as SQL Injection (SQLi) enables the hacker to retrieve data from other users, create and delete data and even cause a denial of service on the database. In the other end, Cross Site Scripting (XSS) is a different attack which makes the application run arbitrary JavaScript code on the client side and possibly stealing sessions, hijacking requests, doing spam and so on. Another vulnerability is Path traversal, that happens when the server doesn't handle files well and retrieve files with name supplied by the user, thus enabling any file from the operating system to be retrieved, such as the shadow file that contains the users and encrypted passwords in that operating system.

Each vulnerability is exploited by different technologies in different ways.

1.2 OBJECTIVES

The following are the objectives of this project

- To Attack signatures are patterns that may indicate malicious traffic, including request types.
- Artificial intelligence algorithms enable behavioral analysis of traffic patterns, using behavioral baselines for various types of traffic to detect anomalies that indicate an attack.
- To involves analyzing the structure of an application, including the typical requests, URLs, values, and permitted data types. This allows the WAF to identify and block potentially malicious requests.

1.3 BACKGROUND AND LITERATURE SURVEY

The Web Application Security Consortium (WASC) is an international group of experts which has been looking at the best-practice security standard for the World Wide Web. There are many other groups working in this area, Thinking Stone being one of them, which has its open-source product called Mod Security. It is an open source WAF acting as a module to an apace web server. Currently, it is the most widely deployed WAF. It can avert already known attacks with the help of configuration files. New attacks cannot be detected. So, the attacks like the Zero Day Attack are not taken into account. We have tried to take these extra features into our product. As far as intrusion detection goes, a lot of clustering algorithms have been studied already. Y-

Means technique initially initializes k clusters randomly. Then the elements are clustered based on the least distance from the cluster center. If there is some empty cluster remaining, then we replace those empty clusters. Instances are then labeled according to the population and k is then adjusted by splitting or merging clusters. It gives greater accuracy than other techniques but is quite costly in terms of time which is an important factor in intrusion detection systems.

Table 1.1 Literature Survey

Sl.No	**Name of article**	**Name of journal**	**Information included**
1.	Machine-Learning-Driven Evolutionary Approach for Testing Web Application Firewalls	Dennis Appelt; Cu D. Nguyen; Annibale Panichella; Lionel C. Briand	This research applies a package namely Reverse Proxy which is used to optimize a web server and Web Application Firewall
2.	Bridging the gap between web application firewalls and web applications	L. Desmet, F. Piessens, W. Joosen and P. Verbaeten	Web applications are the Achilles heel of our current ICT infrastructure
3.	Verified firewall policy transformations for test case generation	A. D. Brucker, L. Brügger, P. Kearney and B. Wolff	Optimization technique for model-based generation of test cases for firewalls
4.	Management of an academic HPC cluster: The UL experience	S. Varrette, P. Bouvry, H. Cartiaux and F. Georgatos	The intensive growth of processing power, data storage and transmission capabilities has revolutionized many aspects of science
5.	Automated pseudo-live testing of firewall configuration enforcement	E. Al-Shaer, A. El-Atawy and T. Samak,	Network security devices such as firewalls and intrusion detection systems are constantly updated in their implementation to accommodate new features

1.4 Need for Web Application Firewall

WAF protects your web apps by filtering, monitoring, and blocking any malicious HTTP/S traffic traveling to the web application, and prevents any unauthorized data from leaving the app. Useful in Company infrastructures

1.5 ORGANIZATION OF THE REPORT

The remaining chapters of the project report are described as follows:

Chapter 2 contains the methodology, design approach, standards used and constraints of the methodology used in the project.

Chapter 3 contains the implementation of K-means clustering. It also explains the communication of clusters.

Chapter 4 gives the cost involved in the implementation of the project and analyses its economic feasibility.

Chapter 5 compiles the results obtained after the project was implemented.

Chapter 6 concludes the report with discussions about the results obtained and their future implications.

CHAPTER 2

WEB APPLICATION FIREWALL

This Chapter describes the methodology, design, standards, calibrations, sampling rate and constraints of Precision Agriculture system.

2.1 Methodology

The idea implemented WAFs can be built into hardware appliances, server-side software plugins, or filter traffic as-a-service. WAF security protects web applications from malicious endpoints and are essentially opposites of proxy servers (i.e., reverse proxies), which protect devices from malicious applications. To ensure security, WAFs intercept and examine all HTTP requests. Bogus traffic is simply blocked or tested with CAPTCHA tests designed to stump harmful bots and computer programs. The fine print of WAF administration is based on security procedures that are built upon customized policies, which should address the top web application security flaws listed by the Open Web Application Security Project (OWASP). The clustering algorithms use the dataset to form clusters and detect intrusions. Almost all of them work on the following two assumptions:

- The normal instances have similar properties and occur close together to form one single cluster while anomalies are far apart from them.
- The number of normal instances is exceedingly large than the number of anomalies i.e., normal instances constitute around 95-98 % of the total data while anomalies constitute the rest.

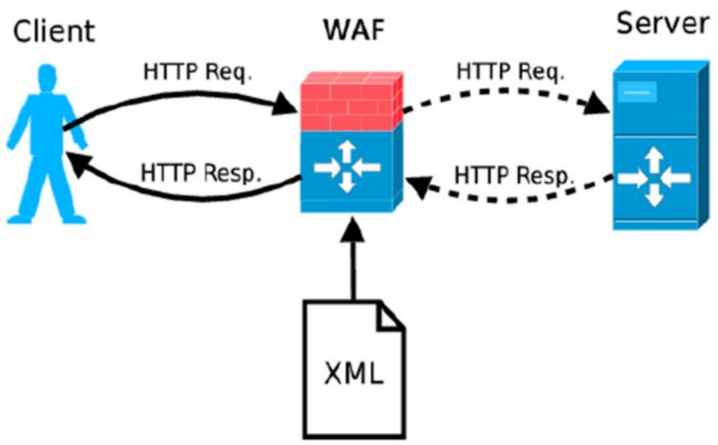


Figure 2.1 Physical Block Diagram

The primary aim of this chapter is to verify the effectiveness of the algorithm proposed through the preprocessing of QR code images. The preprocessing process includes grayscale, filtering, binarization, distortion correction, and perspective projection inverse transformation. The filtering uses an improved adaptive median filter algorithm to filter the image, and the distortion correction uses a distortion correction method based on the BP neural network.

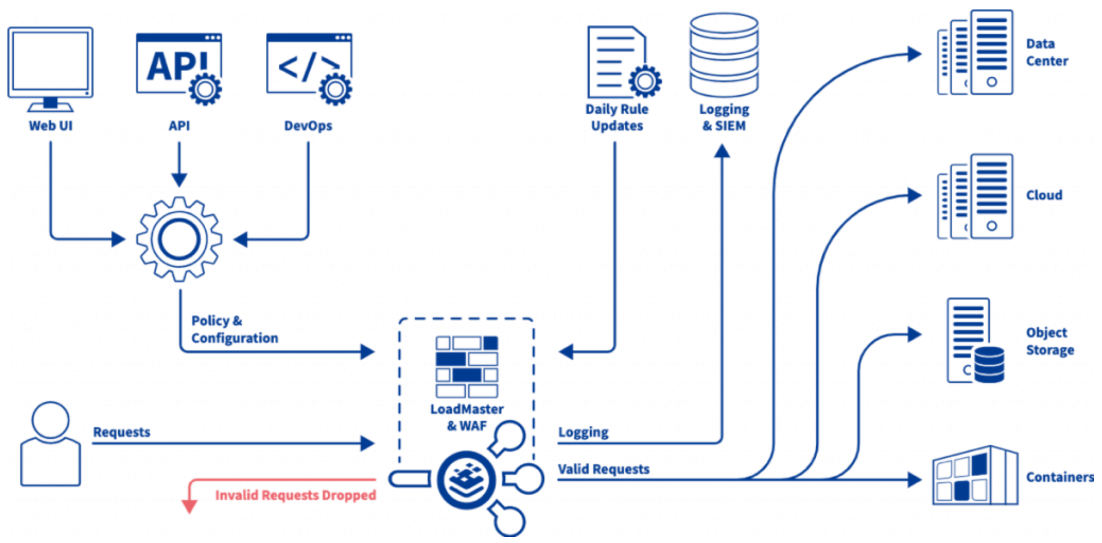


Figure 2.2 External Design Flowchart

A WAF that operates based on a blocklist (negative security model) protects against known attacks. Think of a blocklist WAF as a club bouncer instructed to deny admittance to guests who don’t meet the dress code. Conversely, a WAF based on an allow list (positive security model) only admits traffic that has been pre-approved. This is like the bouncer at an exclusive party, he or she only admits people who are on the list. Both blocklists and allow lists have their advantages and drawbacks, which is why many WAFs offer a hybrid security model, which implements both.

2.2 Design Approach

WAFs can follow either a positive security model, a negative security model, or a combination of both. A positive security model WAF (also known as “allow list”) rejects everything not named as allowed. A negative security model (also known as “deny list”) has a list of banned items and allows everything not on that list.Positive and negative WAF security models have their parts in different application security scenarios. For example:

Positive Security Model

When performing input validation, the positive model dictates that you specify the allowed inputs, as opposed to trying to filter out bad inputs. The benefit of using a positive security model firewall is that new attacks, not anticipated by the developer, will be prevented.

Negative Security Model

The negative model is easier to implement but you'll never be quite sure that you've addressed everything. You'll also end up with a long list of negative signatures to block that has to be maintained. The negative security model approach initially allows all traffic to come through, although as additional restrictions are implemented, security improves. This method can save time for departments that consistently make new network changes, so the network does not continue to be blocked

2.3 Database Configuration of Firewall

In figure 2.3 Firewall configuration involves configuring domain names and Internet Protocol (IP) addresses and completing several other actions to keep firewalls secure. Firewall policy configuration is based on network types called "profiles" that can be set up with security rules to prevent cyber-attacks.

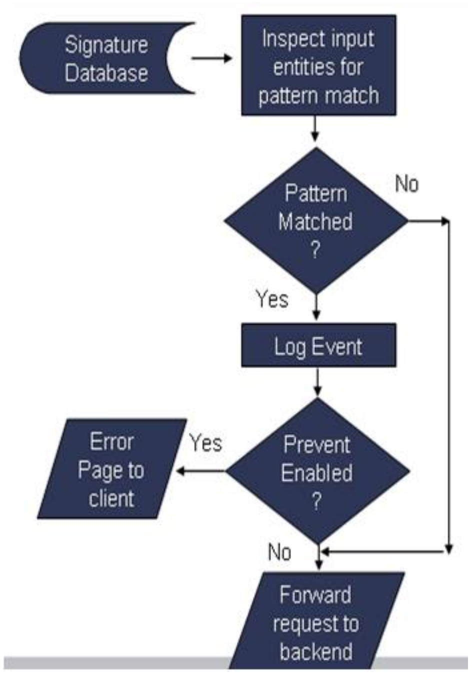


Figure 2.3 Database Configuration Flowchart

- Convolution Neural Nets:
  Convolutional neural networks are composed of multiple layers of artificial neurons. Artificial neurons, a rough imitation of their biological counterparts, are mathematical functions that calculate the weighted sum of multiple inputs and outputs an activation value. When you input an image in a Convent, each layer generates several activation functions that are passed on to the next layer.

- K-Means Clustering:
  k-means clustering is a method of vector quantization, originally from signal processing, that aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean (cluster centres or cluster centroid), serving as a prototype of the cluster.

2.4 ADVANTAGES

- Prevent attacks, including SQL injections, cross-site scripting (XSS) attacks, and distributed denial of service (DDoS) attacks.
- Stop customer data from being compromised, preserving confidence—and their patronage.
- Ensure compliance with regulations like HIPAA and PCI.
- Free up your team’s resources by automatically running security tests and monitoring traffic.

CHAPTER 3

RESULTS AND DISCUSSIONS

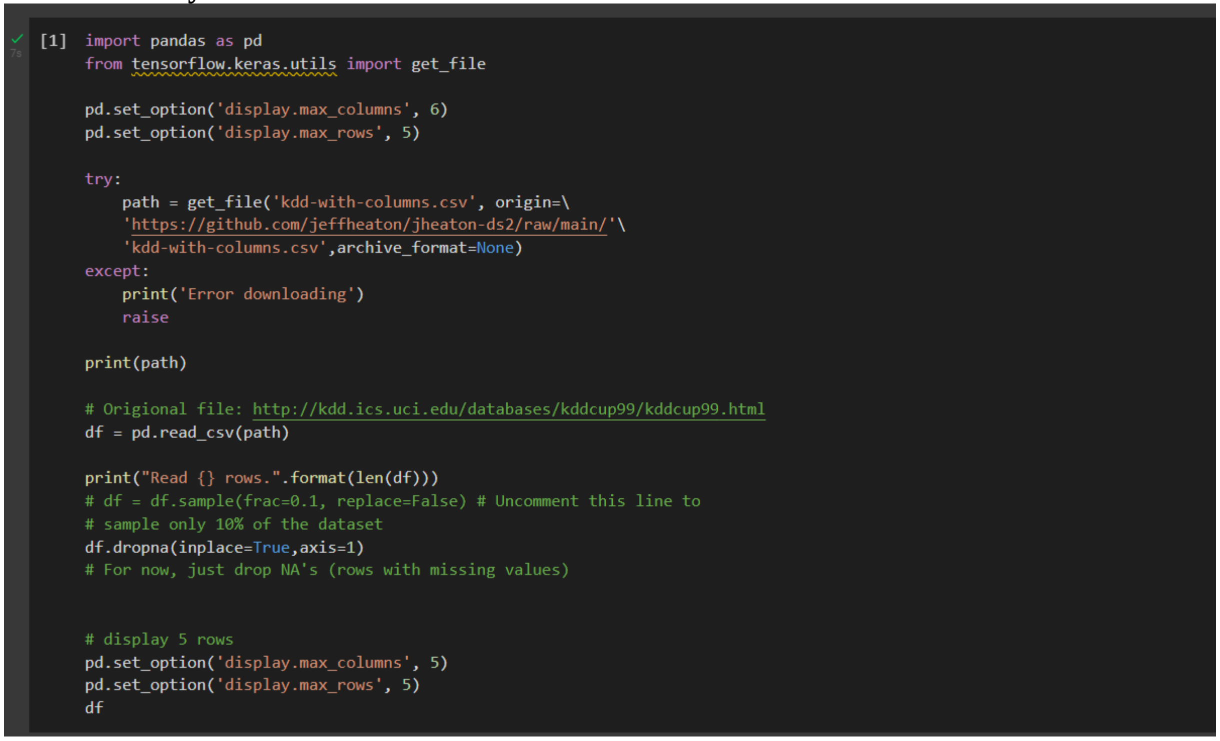
Colab:

Colaboratory, or “Colab” for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing access free of charge to computing resources including GPUs.

Internet application firewalls (WAFs) are usually used to hit upon (and now and again block) web assaults. Many industrial WAFs are to be had, in conjunction with several freely available (typically open supply) alternatives. WAFs can be difficult to personalize for a selected utility, making it hard to run them in “whitelisting mode.” it is commonplace to find WAFs deployed in “blacklisting mode,” making them greater at risk of bypasses and centered assaults. maximum open supply WAFs have a publicly reachable demo software showing the effectiveness of their filtering, and on occasion the WAF's administrative interface as well.

Our web application firewall is an innovative protection system that detects and blocks attacks including the OWASP Top 10, WASC, layer 7 DDoS, and zero-day attacks with pinpoint accuracy. It ensures continuous security for applications, APIs, users, and infrastructure while supporting compliance with security standards including PCI DSS.

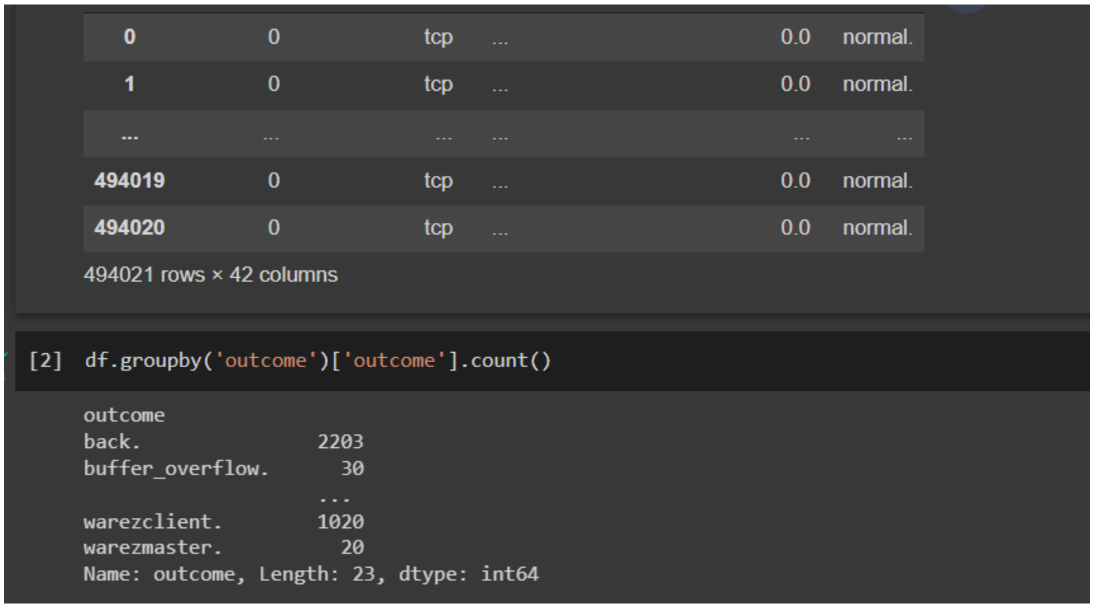
3.1 Anomaly Detection



```
[1] import pandas as pd
    from tensorflow.keras.utils import get_file

    pd.set_option('display.max_columns', 6)
    pd.set_option('display.max_rows', 5)

    try:
        path = get_file('kdd-with-columns.csv', origin=\
        'https://github.com/jeffheaton/jheaton-ds2/raw/main/'\
        'kdd-with-columns.csv',archive_format=None)
    except:
        print('Error downloading')
        raise

    print(path)

    # Origional file: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
    df = pd.read_csv(path)

    print("Read {} rows.".format(len(df)))
    # df = df.sample(frac=0.1, replace=False) # Uncomment this line to
    # sample only 10% of the dataset
    df.dropna(inplace=True,axis=1)
    # For now, just drop NA's (rows with missing values)


    # display 5 rows
    pd.set_option('display.max_columns', 5)
    pd.set_option('display.max_rows', 5)
    df
```

Figure 3.1 Reading the Dataset

The KDD99 dataset contains many columns that define the network state over many time intervals during which a cyber attack might have taken place.The below outcome labelled as “normal” indicated no attack or the the type of attack performed.



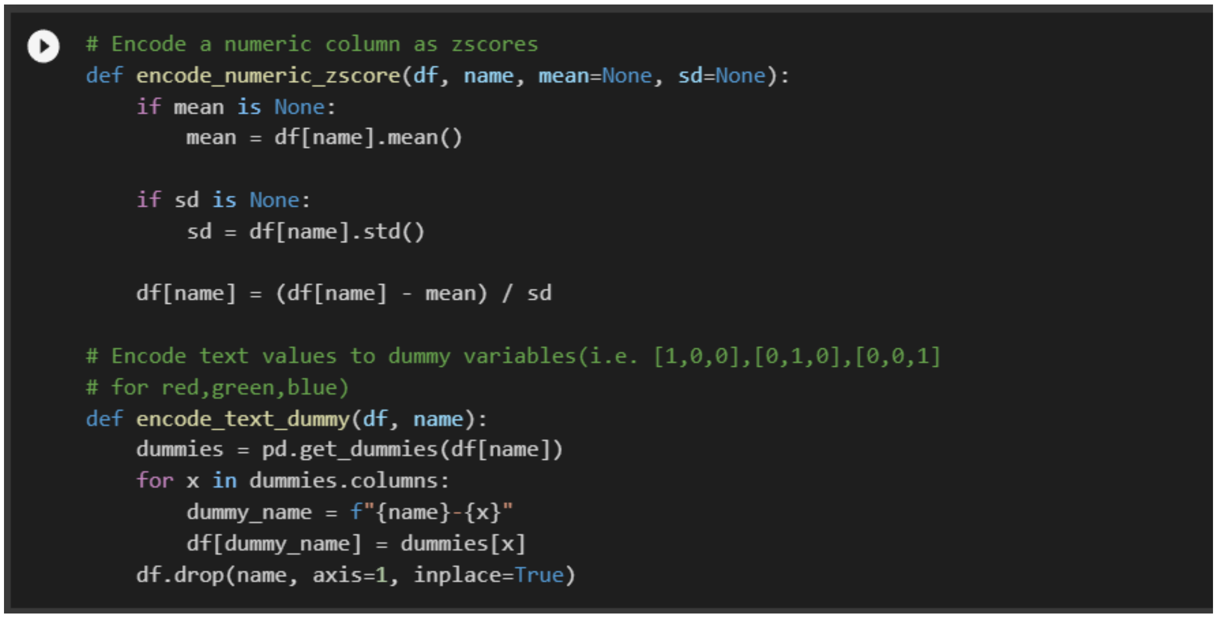
```
     0        0    tcp   ...   0.0   normal.
     1        0    tcp   ...   0.0   normal.
   ...      ...    ...   ...   ...   ...
494019        0    tcp   ...   0.0   normal.
494020        0    tcp   ...   0.0   normal.
494021 rows × 42 columns
```

```
[2] df.groupby('outcome')['outcome'].count()

outcome
back.               2203
buffer_overflow.      30
                    ...
warezclient.        1020
warezmaster.          20
Name: outcome, Length: 23, dtype: int64
```

Figure 3.2 Indication of attack

Preprocessing:

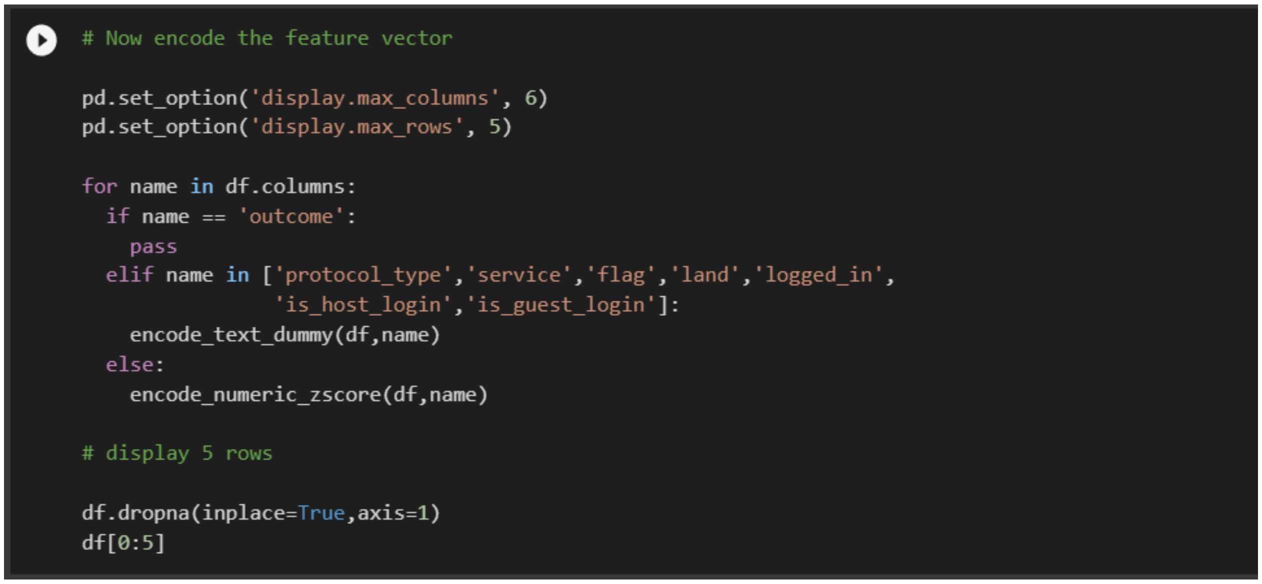
We must perform some preprocessing before we can feed the dataset into the neural network. The below two functions mean and sd assist with the preprocessing. The first function converts numeric columns into z-scorers and the second one replaces categorical values with dummy variables.



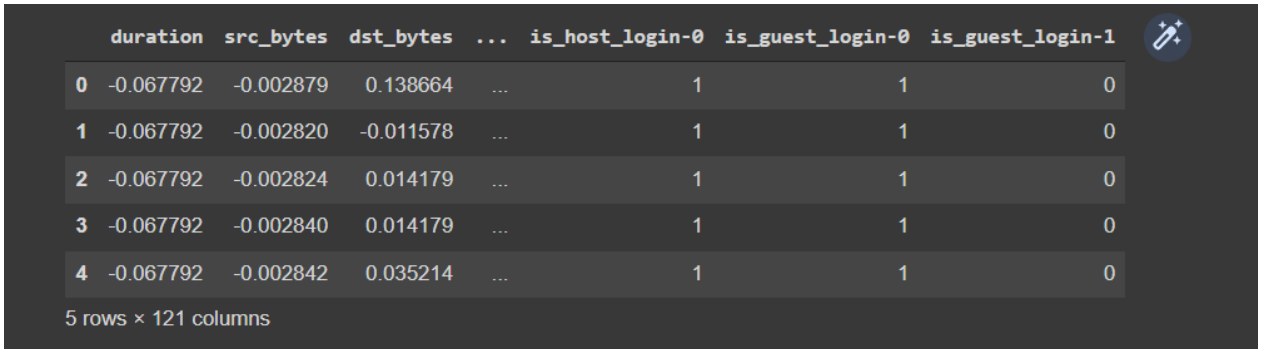
```
# Encode a numeric column as zscores
def encode_numeric_zscore(df, name, mean=None, sd=None):
    if mean is None:
        mean = df[name].mean()

    if sd is None:
        sd = df[name].std()

    df[name] = (df[name] - mean) / sd

# Encode text values to dummy variables(i.e. [1,0,0],[0,1,0],[0,0,1]
# for red,green,blue)
def encode_text_dummy(df, name):
    dummies = pd.get_dummies(df[name])
    for x in dummies.columns:
        dummy_name = f"{name}-{x}"
        df[dummy_name] = dummies[x]
    df.drop(name, axis=1, inplace=True)
```

Figure 3.3 Preprocessing of the data

This code converts all numeric columns to z-scores and all textual columns to dummy variables.



```
# Now encode the feature vector

pd.set_option('display.max_columns', 6)
pd.set_option('display.max_rows', 5)

for name in df.columns:
  if name == 'outcome':
    pass
  elif name in ['protocol_type','service','flag','land','logged_in',
                'is_host_login','is_guest_login']:
    encode_text_dummy(df,name)
  else:
    encode_numeric_zscore(df,name)

# display 5 rows

df.dropna(inplace=True,axis=1)
df[0:5]
```

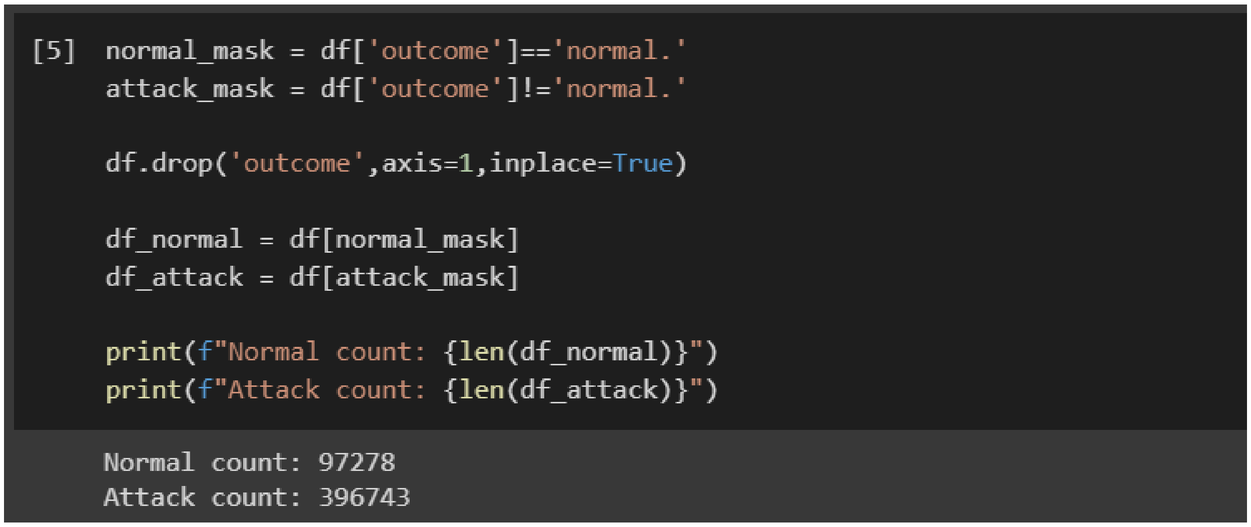


	duration	src_bytes	dst_bytes	...	is_host_login-0	is_guest_login-0	is_guest_login-1
0	-0.067792	-0.002879	0.138664	...	1	1	0
1	-0.067792	-0.002820	-0.011578	...	1	1	0
2	-0.067792	-0.002824	0.014179	...	1	1	0
3	-0.067792	-0.002840	0.014179	...	1	1	0
4	-0.067792	-0.002842	0.035214	...	1	1	0

5 rows × 121 columns

Figure 3.4 Division of the data

To perform anomaly detection, we divide the data into two groups "normal" and the various attacks. The following code divides the data into two dataframes and displays each of these two groups' sizes.

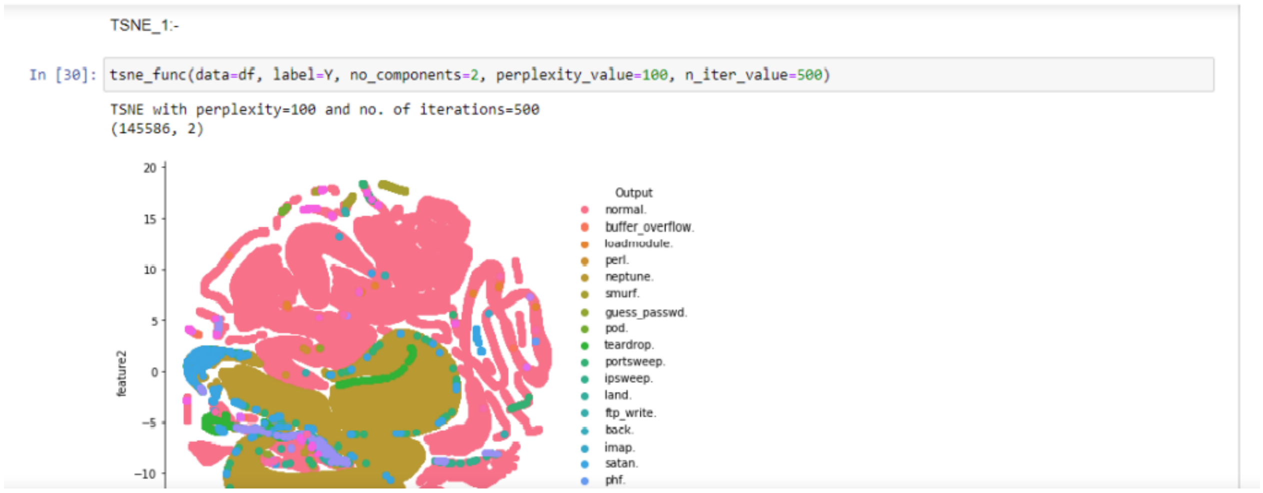


```
[5] normal_mask = df['outcome']=='normal.'
    attack_mask = df['outcome']!='normal.'

    df.drop('outcome',axis=1,inplace=True)

    df_normal = df[normal_mask]
    df_attack = df[attack_mask]

    print(f"Normal count: {len(df_normal)}")
    print(f"Attack count: {len(df_attack)}")
```

```
    Normal count: 97278
    Attack count: 396743
```

Figure 3.5 Type of attack indicated

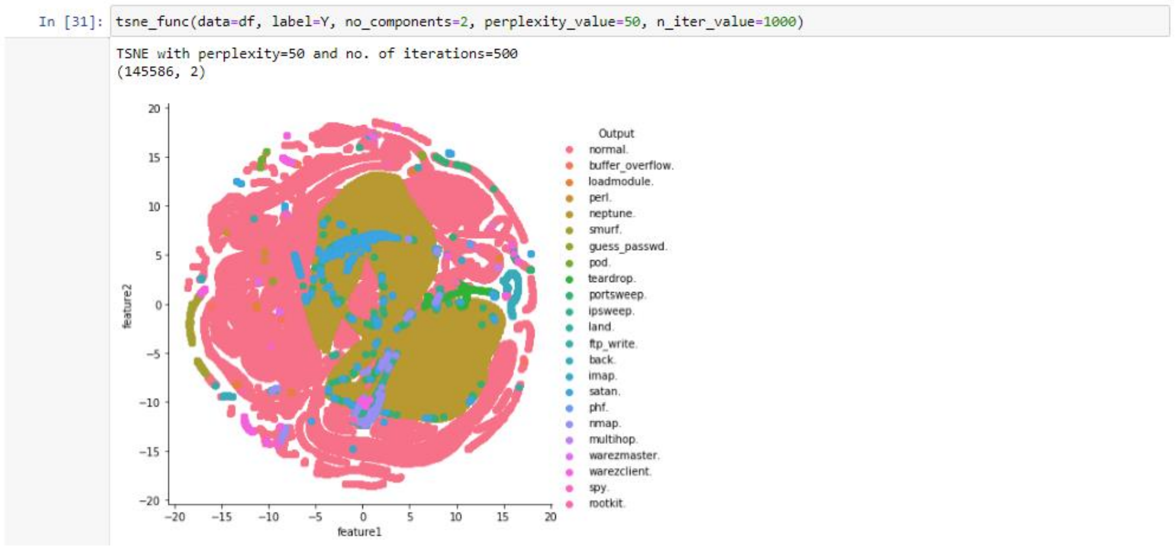
Clustering:



TSNE_1:-

```
In [30]: tsne_func(data=df, label=Y, no_components=2, perplexity_value=100, n_iter_value=500)
```

```
TSNE with perplexity=100 and no. of iterations=500
(145586, 2)
```



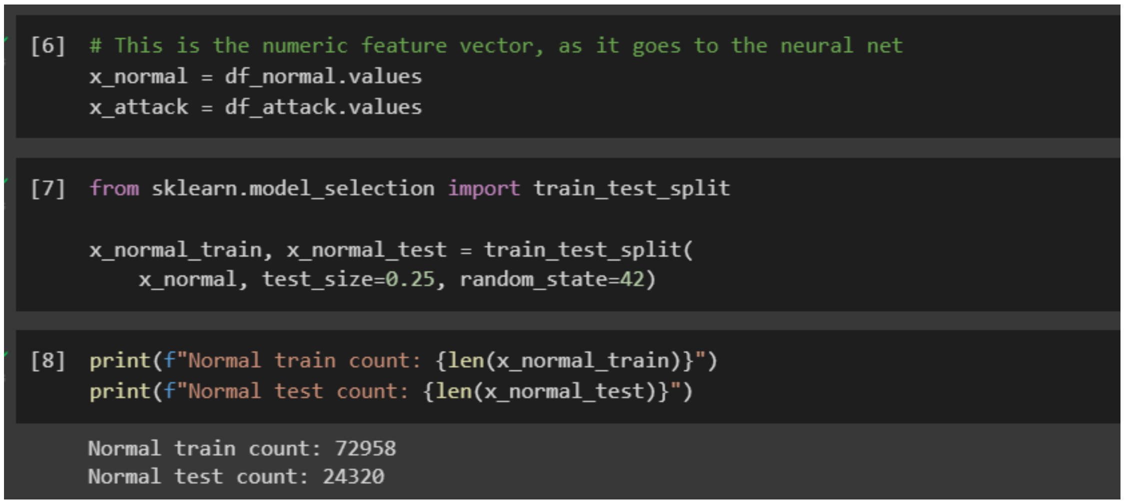
```
In [31]: tsne_func(data=df, label=Y, no_components=2, perplexity_value=50, n_iter_value=1000)

TSNE with perplexity=50 and no. of iterations=500
(145586, 2)
```

Figure 3.6 Clustering

From the above 2 plots, we can conclude that there is no linear separability between any 2 or more categories in the TSNE transformed 2-D space.

Training the Autoencoder

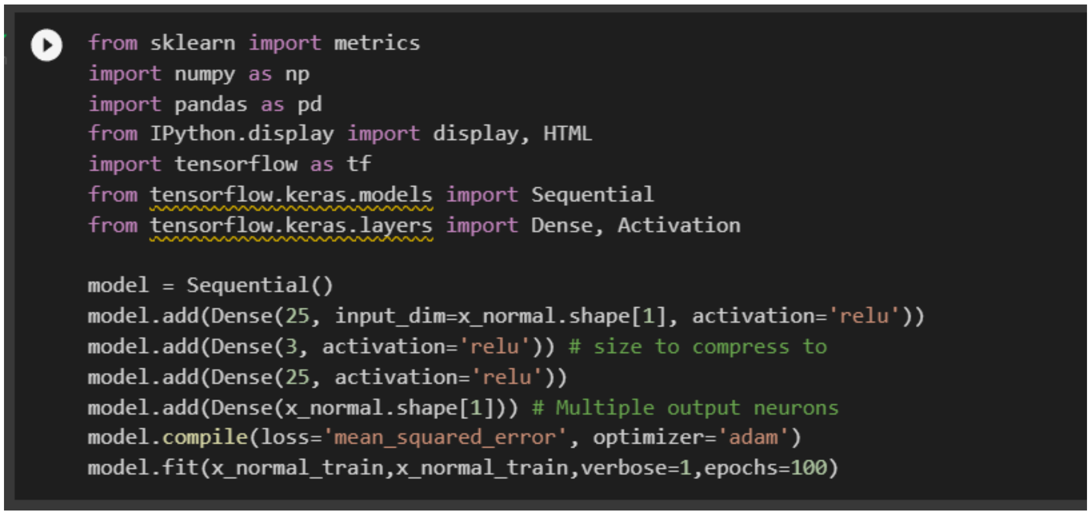
We will train an autoencoder on the normal data and see how well it can detect that the data not flagged as "normal" represents an anomaly.



```
[6]  # This is the numeric feature vector, as it goes to the neural net
     x_normal = df_normal.values
     x_attack = df_attack.values

[7]  from sklearn.model_selection import train_test_split

     x_normal_train, x_normal_test = train_test_split(
         x_normal, test_size=0.25, random_state=42)

[8]  print(f"Normal train count: {len(x_normal_train)}")
     print(f"Normal test count: {len(x_normal_test)}")

     Normal train count: 72958
     Normal test count: 24320
```

Figure 3.7 Training the Autoencoder

We are now ready to train the autoencoder on the normal data. The autoencoder will learn to compress the data to a vector of just three numbers.

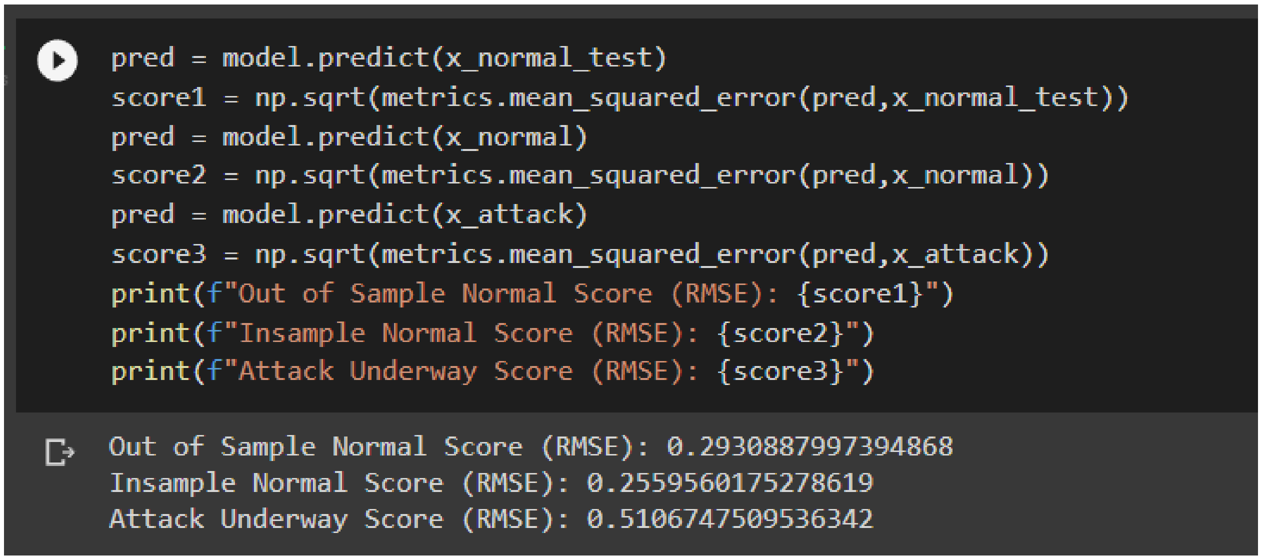


```
from sklearn import metrics
import numpy as np
import pandas as pd
from IPython.display import display, HTML
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation

model = Sequential()
model.add(Dense(25, input_dim=x_normal.shape[1], activation='relu'))
model.add(Dense(3, activation='relu')) # size to compress to
model.add(Dense(25, activation='relu'))
model.add(Dense(x_normal.shape[1])) # Multiple output neurons
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(x_normal_train,x_normal_train,verbose=1,epochs=100)
```

Figure 3.8 Importing Libraries

Detecting an Anomaly

We are now ready to see if the abnormal data registers as an anomaly. The first two scores show the in-sample and out of sample RMSE errors. Both of these two scores are relatively low at around 0.33 because they resulted from normal data. The much higher 0.76 error occurred from the abnormal data. The autoencoder is not as capable of encoding data that represents an attack. This higher error indicates an anomaly.



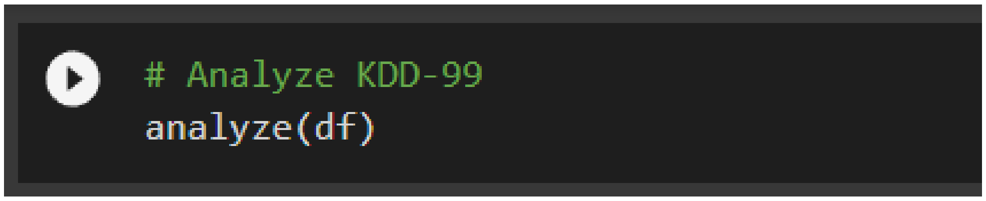
```
pred = model.predict(x_normal_test)
score1 = np.sqrt(metrics.mean_squared_error(pred,x_normal_test))
pred = model.predict(x_normal)
score2 = np.sqrt(metrics.mean_squared_error(pred,x_normal))
pred = model.predict(x_attack)
score3 = np.sqrt(metrics.mean_squared_error(pred,x_attack))
print(f"Out of Sample Normal Score (RMSE): {score1}")
print(f"Insample Normal Score (RMSE): {score2}")
print(f"Attack Underway Score (RMSE): {score3}")
```

```
Out of Sample Normal Score (RMSE): 0.2930887997394868
Insample Normal Score (RMSE): 0.2559560175278619
Attack Underway Score (RMSE): 0.5106747509536342
```

Figure 3.9 Anomaly Detection

3.2 Training an Intrusion Detection System

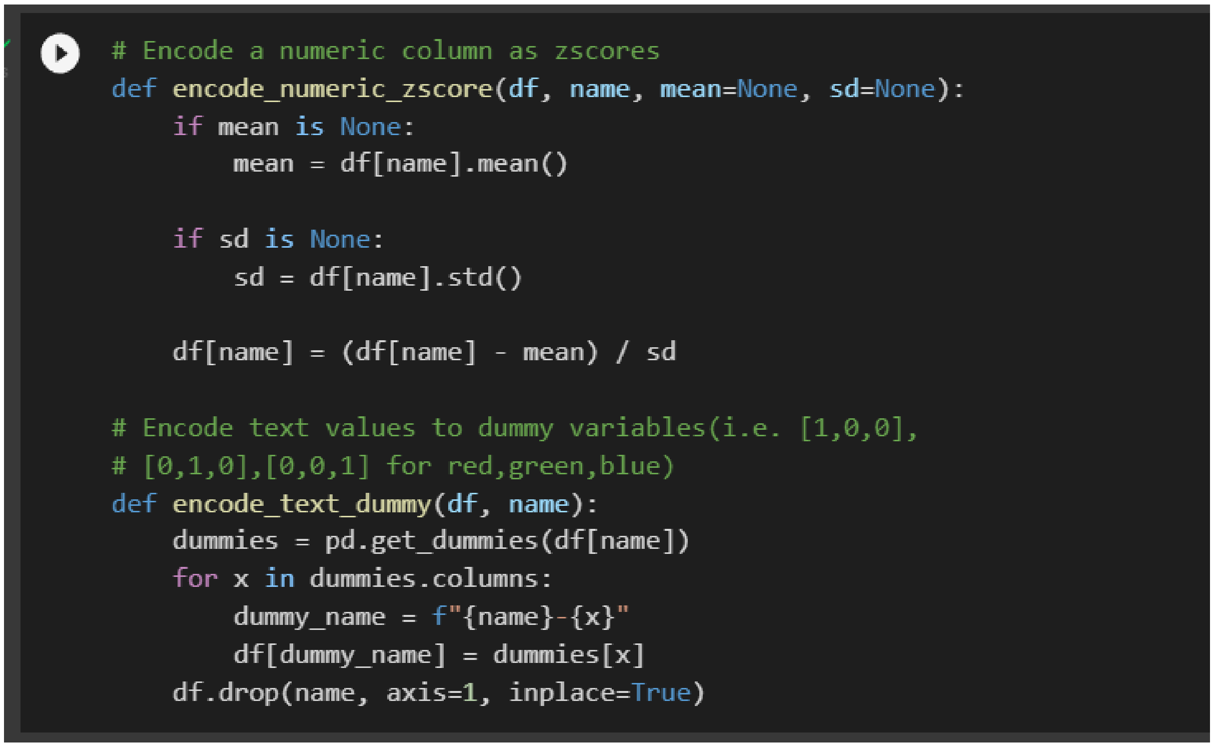
Analyzing the Dataset:



```
# Analyze KDD-99
analyze(df)
```

Figure 3.10 Analyzing the Dataset

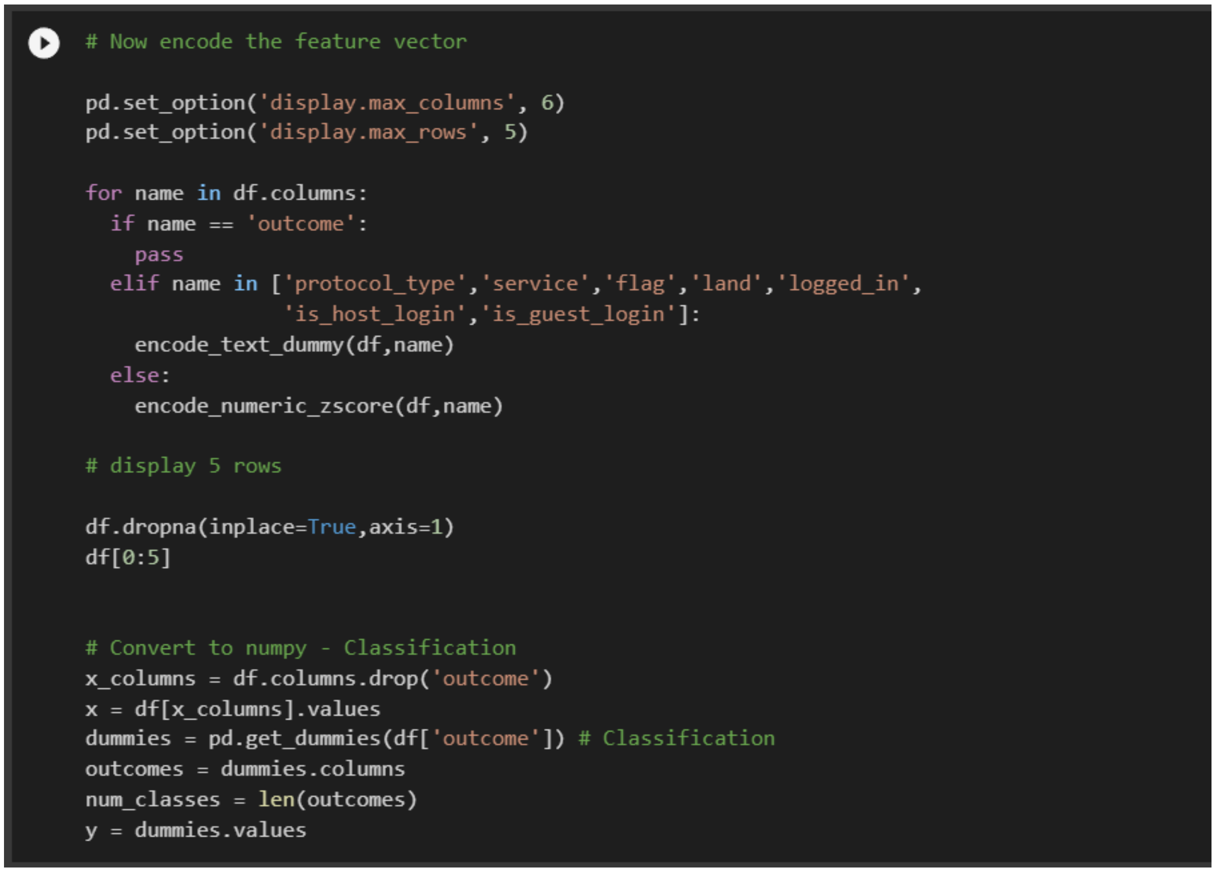
Encode the feature vector:



```
# Encode a numeric column as zscores
def encode_numeric_zscore(df, name, mean=None, sd=None):
    if mean is None:
        mean = df[name].mean()

    if sd is None:
        sd = df[name].std()

    df[name] = (df[name] - mean) / sd

# Encode text values to dummy variables(i.e. [1,0,0],
# [0,1,0],[0,0,1] for red,green,blue)
def encode_text_dummy(df, name):
    dummies = pd.get_dummies(df[name])
    for x in dummies.columns:
        dummy_name = f"{name}-{x}"
        df[dummy_name] = dummies[x]
    df.drop(name, axis=1, inplace=True)
```

Figure 3.11 Encoding the feature Vector

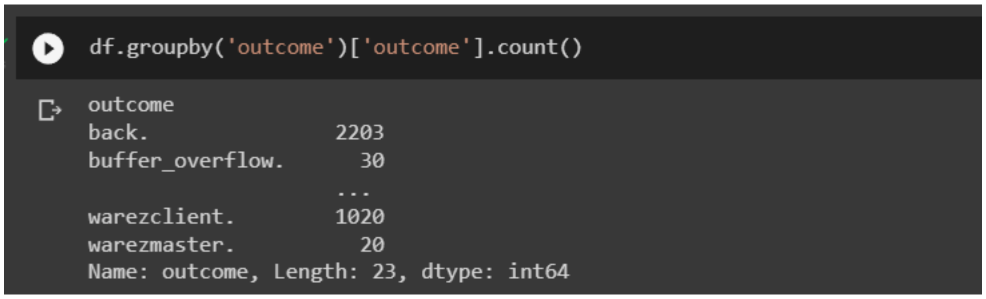
Again, just as we did for anomaly detection, we preprocess the data set. We convert all numeric values to Z-Score, and we translate all categorical to dummy variables.



```
# Now encode the feature vector

pd.set_option('display.max_columns', 6)
pd.set_option('display.max_rows', 5)

for name in df.columns:
  if name == 'outcome':
    pass
  elif name in ['protocol_type','service','flag','land','logged_in',
                'is_host_login','is_guest_login']:
    encode_text_dummy(df,name)
  else:
    encode_numeric_zscore(df,name)

# display 5 rows

df.dropna(inplace=True,axis=1)
df[0:5]


# Convert to numpy - Classification
x_columns = df.columns.drop('outcome')
x = df[x_columns].values
dummies = pd.get_dummies(df['outcome']) # Classification
outcomes = dummies.columns
num_classes = len(outcomes)
y = dummies.values
```

Figure 3.12 Conversion of data into Z-score and dummy variable

We will attempt to predict what type of attack is underway.



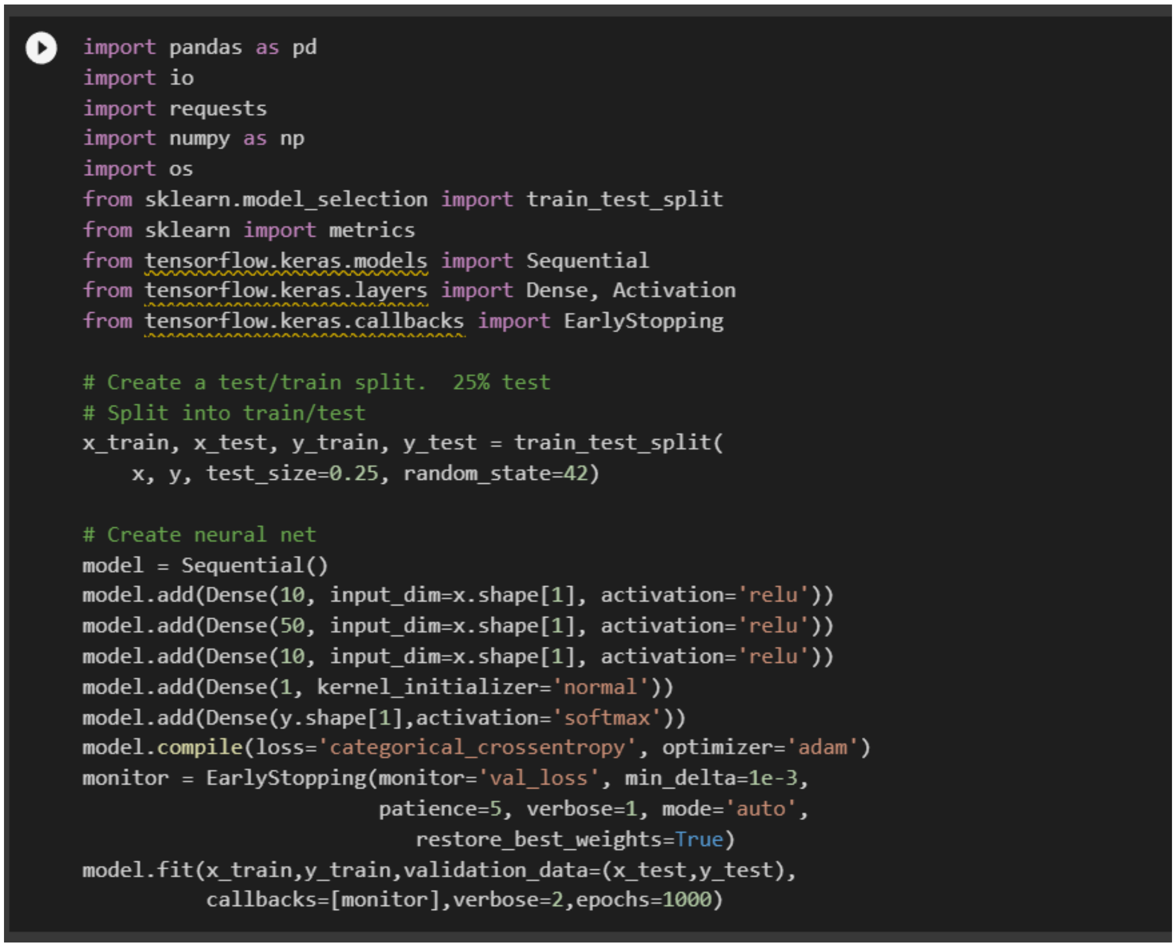
```
df.groupby('outcome')['outcome'].count()
```

```
outcome
back.                2203
buffer_overflow.       30
                     ...
warezclient.         1020
warezmaster.           20
Name: outcome, Length: 23, dtype: int64
```

Figure 3.13 Grouping of dataset

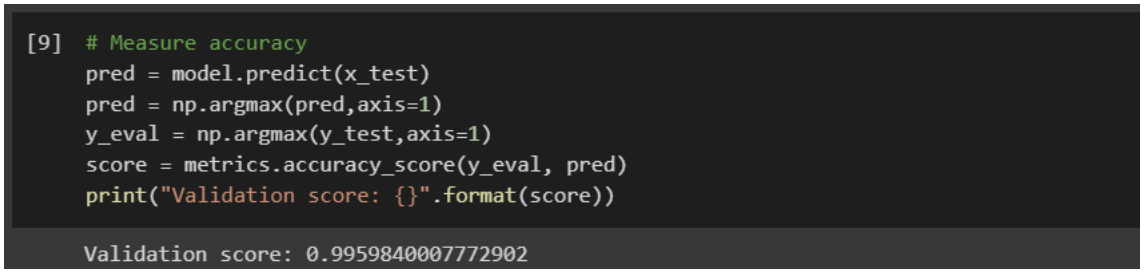
Train the Neural Network:

We now train the neural network to classify the different KDD99 outcomes. The code provided here implements a relatively simple neural with two hidden layers. We train it with the provided KDD99 data.



```
import pandas as pd
import io
import requests
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn import metrics
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping

# Create a test/train split.  25% test
# Split into train/test
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.25, random_state=42)

# Create neural net
model = Sequential()
model.add(Dense(10, input_dim=x.shape[1], activation='relu'))
model.add(Dense(50, input_dim=x.shape[1], activation='relu'))
model.add(Dense(10, input_dim=x.shape[1], activation='relu'))
model.add(Dense(1, kernel_initializer='normal'))
model.add(Dense(y.shape[1],activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3,
                        patience=5, verbose=1, mode='auto',
                           restore_best_weights=True)
model.fit(x_train,y_train,validation_data=(x_test,y_test),
          callbacks=[monitor],verbose=2,epochs=1000)
```

Figure 3.14 Training the Neural Network



```
[9]  # Measure accuracy
     pred = model.predict(x_test)
     pred = np.argmax(pred,axis=1)
     y_eval = np.argmax(y_test,axis=1)
     score = metrics.accuracy_score(y_eval, pred)
     print("Validation score: {}".format(score))

     Validation score: 0.9959840007772902
```

Figure 3.15 Accuracy Measurement

We can now evaluate the neural network. As we can see, the neural network achieves a 99% accuracy rate.

Summary

Our web application firewall is an innovative protection system that detects and blocks attacks including the OWASP Top 10, WASC, layer 7 DDoS, and zero-day attacks with pinpoint accuracy. It ensures continuous security for applications, APIs, users, and infrastructure while supporting compliance with security standards including PCI DSS.

CHAPTER 4

CONCLUSION AND FUTURE WORK

4.1 Conclusion

Payloads sent to web applications can be malicious and crafted by hackers to attack applications. The protection of those applications usually occurs by the usage of a web application firewall (WAF), that can be exploited by hackers. The proposal of this work was to use a learning base approach to identify new payloads. By achieving a high accuracy on classifying new payloads, the model can be used in conjunction with the WAF to evade malicious attacks on web applications, by avoiding several anti-WAF techniques. The model can also be used as an application middleware, capable of accepting or rejecting payloads based on malicious payloads or application logic.

4.2 Future work

The next goal is to add new payloads classes and different vulnerabilities that target specific web servers' versions to retrain the model. Also develop an architecture that enables the model to retrain from time to time, keeping up to date to new emerging web application attacks by plugging the model to a WAF to collect and analyze its performance on a real scenario, and also, measure the model accuracy bench-marking against other machine learning models.

APPENDIX

1. Anomaly Detection

```
import pandas as pd
from tensorflow.keras.utils import get_file

pd.set_option('display.max_columns', 6)
pd.set_option('display.max_rows', 5)

try:
    path = get_file('kdd-with-columns.csv', origin=\
    'https://github.com/jeffheaton/jheaton-ds2/raw/main/'\
    'kdd-with-columns.csv',archive_format=None)
except:
    print('Error downloading')
    raise

print(path)

# Origional file: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
df = pd.read_csv(path)

print("Read {} rows.".format(len(df)))
# df = df.sample(frac=0.1, replace=False) # Uncomment this line to
# sample only 10% of the dataset
df.dropna(inplace=True,axis=1)
# For now, just drop NA's (rows with missing values)


# display 5 rows
pd.set_option('display.max_columns', 5)
pd.set_option('display.max_rows', 5)
df
df.groupby('outcome')['outcome'].count()
# Encode a numeric column as zscores
def encode_numeric_zscore(df, name, mean=None, sd=None):
    if mean is None:
        mean = df[name].mean()

    if sd is None:
        sd = df[name].std()

    df[name] = (df[name] - mean) / sd

# Encode text values to dummy variables(i.e. [1,0,0],[0,1,0],[0,0,1]
# for red,green,blue)
```

```
def encode_text_dummy(df, name):
    dummies = pd.get_dummies(df[name])
    for x in dummies.columns:
        dummy_name = f"{name}-{x}"
        df[dummy_name] = dummies[x]
    df.drop(name, axis=1, inplace=True)
# Now encode the feature vector

pd.set_option('display.max_columns', 6)
pd.set_option('display.max_rows', 5)

for name in df.columns:
  if name == 'outcome':
    pass
  elif name in ['protocol_type','service','flag','land','logged_in',
                'is_host_login','is_guest_login']:
    encode_text_dummy(df,name)
  else:
    encode_numeric_zscore(df,name)

# display 5 rows

df.dropna(inplace=True,axis=1)
df[0:5]
normal_mask = df['outcome']=='normal.'
attack_mask = df['outcome']!='normal.'

df.drop('outcome',axis=1,inplace=True)

df_normal = df[normal_mask]
df_attack = df[attack_mask]

print(f"Normal count: {len(df_normal)}")
print(f"Attack count: {len(df_attack)}")
# This is the numeric feature vector, as it goes to the neural net
x_normal = df_normal.values
x_attack = df_attack.values
from sklearn.model_selection import train_test_split

x_normal_train, x_normal_test = train_test_split(
    x_normal, test_size=0.25, random_state=42)
print(f"Normal train count: {len(x_normal_train)}")
print(f"Normal test count: {len(x_normal_test)}")
from sklearn import metrics
import numpy as np
import pandas as pd
from IPython.display import display, HTML
```

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation

model = Sequential()
model.add(Dense(25, input_dim=x_normal.shape[1], activation='relu'))
model.add(Dense(3, activation='relu')) # size to compress to
model.add(Dense(25, activation='relu'))
model.add(Dense(x_normal.shape[1])) # Multiple output neurons
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(x_normal_train,x_normal_train,verbose=1,epochs=100)
pred = model.predict(x_normal_test)
score1 = np.sqrt(metrics.mean_squared_error(pred,x_normal_test))
pred = model.predict(x_normal)
score2 = np.sqrt(metrics.mean_squared_error(pred,x_normal))
pred = model.predict(x_attack)
score3 = np.sqrt(metrics.mean_squared_error(pred,x_attack))
print(f"Out of Sample Normal Score (RMSE): {score1}")
print(f"Insample Normal Score (RMSE): {score2}")
print(f"Attack Underway Score (RMSE): {score3}")
```

2. Training an Intrusion Detection System

```
import pandas as pd
from tensorflow.keras.utils import get_file

pd.set_option('display.max_columns', 6)
pd.set_option('display.max_rows', 5)

try:
    path = get_file('kdd-with-columns.csv', origin=\
    'https://github.com/jeffheaton/jheaton-ds2/raw/main/'\
    'kdd-with-columns.csv',archive_format=None)
except:
    print('Error downloading')
    raise

print(path)

# Origional file: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
df = pd.read_csv(path)

print("Read {} rows.".format(len(df)))
# df = df.sample(frac=0.1, replace=False) # Uncomment this line to
# sample only 10% of the dataset
df.dropna(inplace=True,axis=1)
# For now, just drop NA's (rows with missing values)


# display 5 rows
pd.set_option('display.max_columns', 5)
pd.set_option('display.max_rows', 5)
df
```

```
import pandas as pd
import os
import numpy as np
from sklearn import metrics
from scipy.stats import zscore

def expand_categories(values):
    result = []
    s = values.value_counts()
    t = float(len(values))
    for v in s.index:
        result.append("{}:{}%".format(v,round(100*(s[v]/t),2)))
    return "[{}]".format(",".join(result))
```

```
def analyze(df):
    print()
    cols = df.columns.values
    total = float(len(df))

    print("{} rows".format(int(total)))
    for col in cols:
        uniques = df[col].unique()
        unique_count = len(uniques)
        if unique_count>100:
            print("** {}:{} ({}%)".format(col,unique_count,\
                int(((unique_count)/total)*100)))
        else:
            print("** {}:{}".format(col,expand_categories(df[col])))
            expand_categories(df[col])
# Analyze KDD-99
analyze(df)
# Encode a numeric column as zscores
def encode_numeric_zscore(df, name, mean=None, sd=None):
    if mean is None:
        mean = df[name].mean()

    if sd is None:
        sd = df[name].std()

    df[name] = (df[name] - mean) / sd

# Encode text values to dummy variables(i.e. [1,0,0],
# [0,1,0],[0,0,1] for red,green,blue)
def encode_text_dummy(df, name):
    dummies = pd.get_dummies(df[name])
    for x in dummies.columns:
        dummy_name = f"{name}-{x}"
        df[dummy_name] = dummies[x]
    df.drop(name, axis=1, inplace=True)
# Now encode the feature vector

pd.set_option('display.max_columns', 6)
pd.set_option('display.max_rows', 5)

for name in df.columns:
  if name == 'outcome':
    pass
  elif name in ['protocol_type','service','flag','land','logged_in',
                'is_host_login','is_guest_login']:
    encode_text_dummy(df,name)
  else:
    encode_numeric_zscore(df,name)

# display 5 rows

df.dropna(inplace=True,axis=1)
df[0:5]


# Convert to numpy - Classification
x_columns = df.columns.drop('outcome')
x = df[x_columns].values
dummies = pd.get_dummies(df['outcome']) # Classification
outcomes = dummies.columns
num_classes = len(outcomes)
y = dummies.values
df.groupby('outcome')['outcome'].count()
import pandas as pd
import io
import requests
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn import metrics
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping

# Create a test/train split.  25% test
# Split into train/test
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.25, random_state=42)

# Create neural net
model = Sequential()
model.add(Dense(10, input_dim=x.shape[1], activation='relu'))
model.add(Dense(50, input_dim=x.shape[1], activation='relu'))
model.add(Dense(10, input_dim=x.shape[1], activation='relu'))
model.add(Dense(1, kernel_initializer='normal'))
model.add(Dense(y.shape[1],activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3,
                        patience=5, verbose=1, mode='auto',
                          restore_best_weights=True)
model.fit(x_train,y_train,validation_data=(x_test,y_test),
          callbacks=[monitor],verbose=2,epochs=1000)
# Measure accuracy
pred = model.predict(x_test)
pred = np.argmax(pred,axis=1)
y_eval = np.argmax(y_test,axis=1)
score = metrics.accuracy_score(y_eval, pred)
print("Validation score: {}".format(score))
```

REFERENCES

[1] Torrano-Gimenez C, Hai TN, Alvarez G et al (2014) Applying feature selection to payloadbased Web Application Firewalls. In: Third international workshop on security and communication networks. IEEE, pp 75–81.

[2] Feng F, Shao C, Liu D (2012) Design and implementation of coldfusion-based web application firewall. In: International conference on computer science & service system. IEEE, pp 659–662

[3] Tekerek A, Gemci C, Bay OF (2015) Development of a hybrid web application firewall to prevent web-based attacks. In: IEEE international conference on application of information and communication technologies. IEEE, pp 1–4

[4] Park H (2015) Advanced security configuration options for SAS® 9.4 web applications and mobile devices.

[5] Liu Z (2014) Analysis on firewall technologies of web-based application. Inf Res 2014

[6] Banjo I (2016) System and method for event-related content discovery, curation, and presentation

[7] Epp N, Funk R, Cappo C (2017) Anomaly-based web application firewall using HTTPspecific features and one-class SVM. In: Workshop Regional de Segurança da Informação e de Sistemas Computacionais

[8] Hadi, A., Al-Bahadili, H.: A hybrid port-knocking technique for host authentication. IGIGlobal Knowledge Disseminator (2012)

[9] Gibbons, K., O'Raw, J., Curran, K.: Security evaluation of the OAuth 2.0 framework. Inf. Manage. Comput. Secur. 22(3), December 2014.

[10] Internet Engineering Task Force (IETF): The OAuth 2.0 Authorization Framework.