National College of Ireland

Configuration Manual

MSc Research Project
MSc in Cybersecurity

Kishore Hariram
Student ID: 21115737

School of Computing
National College of Ireland

Supervisor: Dr. Vanessa Ayala-Rivera

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Kishore Hariram		
Student ID:	21115737		
Programme:	MSc in Cybersecurity	**Year:**	2022
Module:	MSc Research Project/Internship		
Lecturer:	Dr. Vanessa Ayala-Rivera		
Submission Due Date:	01/02/2023		
Project Title:	Detection of Clickjacking using the Convolutional Neural Network		
Word Count:	740	**Page Count:**	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Kishore Hariram

Date: 29/01/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

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

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

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

Configuration Manual

Kishore Hariram
Student ID: 21115737

1 Introduction

The configuration manual contains information about all the technologies and tools used to implement the whole research project. Software and all the technologies used in the research are described in Section 2. Section 3 describes step by step procedure of implementation, execution, and output of the whole research project along with the source code.

2 Technologies and Software

Your second section. To create the static web application, HTML was used which was embedded with iframes.
Python of version 3.9.5 was used for the development of machine learning and detection.

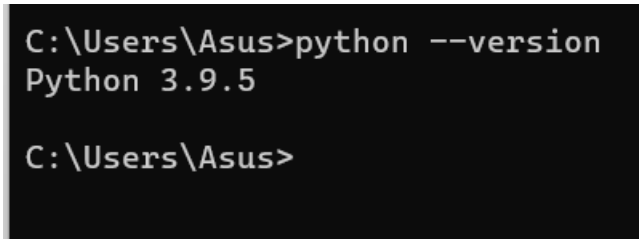


Figure 1: Version of python

Jupyter notebook is used to develop all the machine learning and iframe detection modules. Jupyter notebook is an open-source tool that supports both python programming languages and HTML components. This produces an interactive output for the user.

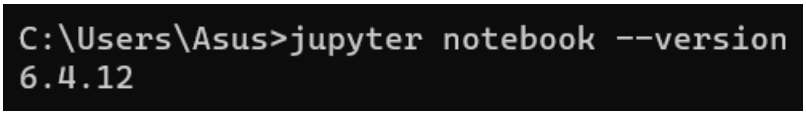
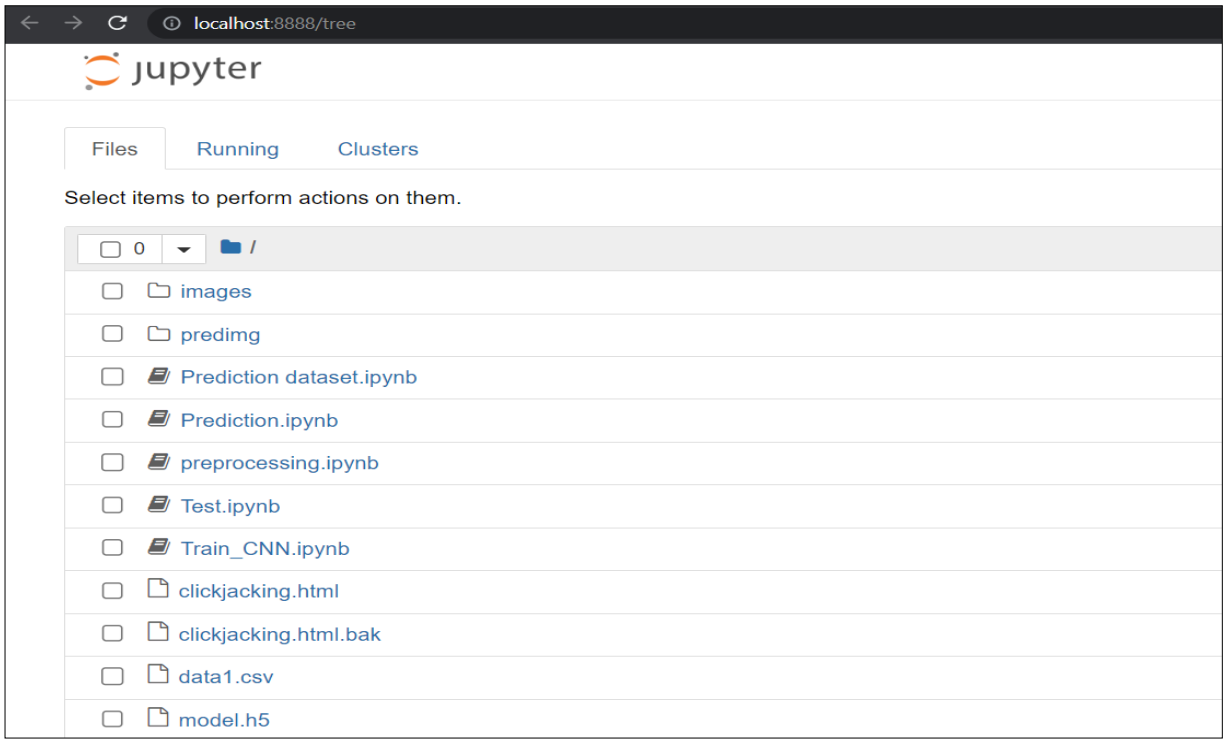


Figure 2: Jupyter notebook version

3 Implementation

Your third section. Change the header and label to something appropriate.

- Step 1: Install the Jupyter notebook into the system and launch Jupyter notebook for the folder having all the source code.



- Step 2: Initially the dataset should be pre-processed.
- Step 3: Import the necessary libraries such as numpy, pandas, tensorflow, time, request, BeautifulSoup, whois and etc.

```
from os import listdir
import numpy as np
import pandas as pd
import tensorflow as tf
import cv2
from tensorflow.keras.models import load_model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D,BatchNormalization
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from bs4 import BeautifulSoup
import requests
import datetime
import whois
import dns.resolver
from tld import get_tld
from datetime import datetime
import imageio as im
import pyautogui
import time
import webbrowser
```

- Step 4: The dataset is loaded and performed the pre-processed.

```
data = pd.read_csv('data1.csv')
data.head()
```

Figure 3: Loading dataset in pre-processing

```
X = data.drop(['phishing'], axis = 1)
y = data['phishing']
X.head()
```

```
X = pd.get_dummies(X)
X.head()
X.info()
```

- Step 5: Once pre-processing is done, the data is normalized which removes null and void values present in the dataset.

```
df=data.corr()
```

```
def min_max_scaling(df):
    df_norm = df.copy()
    for column in df_norm.columns:
        df_norm[column] = (df_norm[column] - df_norm[column].min()) / (df_norm[column].max() - df_norm[column].min())
    return df_norm

X_normalized = min_max_scaling(X)
X_normalized.head()
```

```
no = X_normalized.shape[0]
dummy_val = np.zeros(no)
for i in range(3):
    name = 'dummy' + str(i)
    X_normalized[name] = dummy_val
X_normalized
```

Figure 4: Data Normalization

- Step 6: The dataset is further split into test and train datasets, where the dataset is divides into 40% for testing and 60% for training the model.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_normalized,y,test_size=.40)
```

```
X_train
```

Figure 5: Dataset split

- Step 7: The data is converted into images which will be trained by the CNN model.

```
import imageio as im

def data_to_img(x,y,type_of_data):
    len_of_rows = x.shape[0]
    for i in range(0,len_of_rows):
        temp = np.array(x.iloc[i])
        temp = temp.reshape(10,10)
        filename = 'images/'+type_of_data+'/'+str(y.iloc[i])+'/img_'+str(i)+'.jpg'
        im.imwrite(filename, temp)
```

```
data_to_img(X_train,y_train,'train')
data_to_img(X_test,y_test,'test')
```

Figure 6: Image conversion

- Step 8: The images are further trained using the activation function=’relu’

```
model = Sequential()
model.add(Conv2D(32,(5,5), input_shape=(50,50,1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(filters=32, kernel_size=(2, 2)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())

model.add(Conv2D(filters=64, kernel_size=(2, 2)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.1))

model.add(Conv2D(filters=64, kernel_size=(2, 2)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(units=280, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

Figure 7: Training the data

- Step 9: Once the images are trained and saved as a model file.

```
history = model.fit(X_train, y_train, epochs=40, batch_size=200, verbose=2)
model.save('model.h5')
```

Figure 8: Model file created

- Step 10: Once all set up, the iframes are embedded in the website.

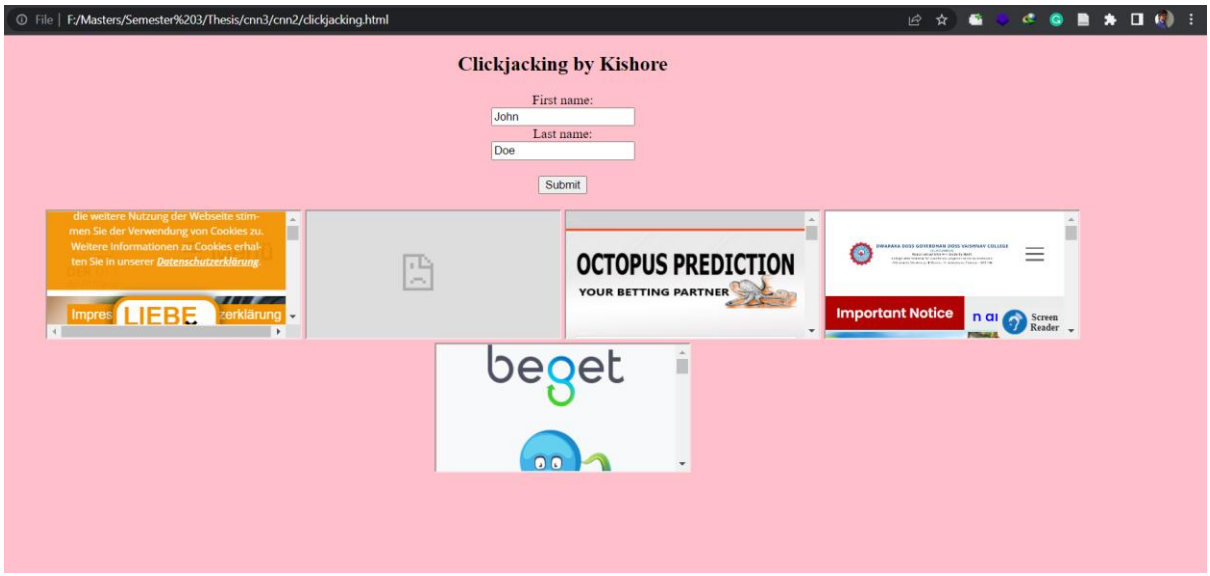


Figure 9: Webpage with malicious iframes

- Step 11: Once the prediction code is executed, Window is opened and highlighted with the malicious iframes.

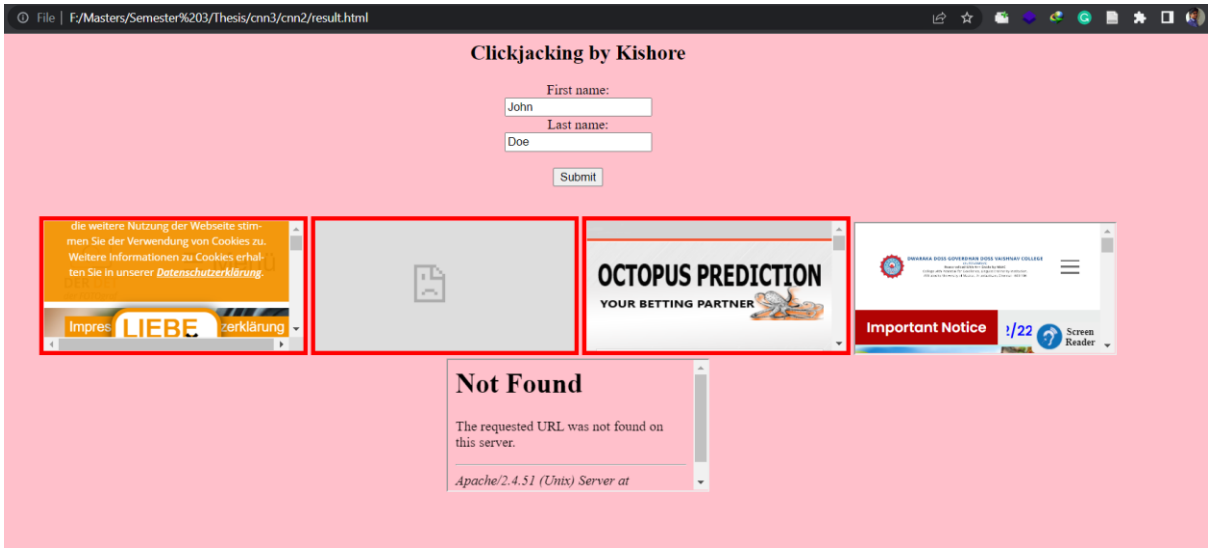


Figure 10: Clickjack able iframes are highlighted

Source Codes:

- For the prediction, the following code shows the extraction of URL from the iframes using BeautifulSoup scrapping method.

```
def parse(f):
    p=BeautifulSoup(f, "html.parser")
    Url=[]
    for i in p.find_all("iframe"):
        url = i.attrs['src']
        Url.append(url)
    print(Url)
    return Url
```

Figure 11: Web scrapping using BeautifulSoup

- The feature extraction of the URLs are show in below pictures.

```
def url(k):
    l=[]
    for i in k:
        i=str(i)
        d={}
        d["qty_dot_url"]=i.count('.')
        d['qty_hyphen_url']=i.count('-')
        d['qty_underline_url']=i.count('_')
        d['qty_slash_url']=i.count('/')
        d['qty_questionmark_url']=i.count('?')
        d['qty_equal_url']=i.count('=')
        d['qty_at_url']=i.count('@')
        d['qty_and_url']=i.count('&')
        d['qty_exclamation_url']=i.count('!')
        d['qty_space_url']=i.count(' ')
        d['qty_tilde_url']=i.count('~')
        d['qty_comma_url']=i.count(',')
        d['qty_plus_url']=i.count('+')
        d['qty_asterik_url']=i.count('*')
        d['qty_hashtag_url']=i.count('#')
        d['qty_dollar_url']=i.count('$')
        d['qty_percent_url']=i.count('%')
        d['length_url']=len(i)
        l.append(d)
    df=pd.DataFrame(l)
    return df
```

```
def domain(g):
        l=[]
        for i in g:
            i=str(i)
            d={}
            d["qty_dot_domain"]=i.count('.')
            d['qty_hyphen_domain']=i.count('-')
            d['qty_underline_domain']=i.count('_')
            d['qty_slash_domain']=i.count('/')
            d['qty_questionmark_domain']=i.count('?')
            d['qty_equal_domain']=i.count('=')
            d['qty_at_domain']=i.count('@')
            d['qty_and_domain']=i.count('&')
            d['qty_exclamation_domain']=i.count('!')
            d['qty_space_domain']=i.count(' ')
            d['qty_tilde_domain']=i.count('~')
            d['qty_comma_domain']=i.count(',')
            d['qty_plus_domain']=i.count('+')
            d['qty_asterik_domain']=i.count('*')
            d['qty_hashtag_domain']=i.count('#')
            d['qty_dollar_domain']=i.count('$')
            d['qty_percent_domain']=i.count('%')
            d['qty_vowels_domain']=countVowels(i)
            d['domain_length']=len(i)
            l.append(d)
        df=pd.DataFrame(l)
        return df
```

```
def Directory(h):
        l=[]
        for i in h:
            i=str(i)
            d={}
            d["qty_dot_directory"]=i.count('.')
            d['qty_hyphen_directory']=i.count('-')
            d['qty_underline_directory']=i.count('_')
            d['qty_slash_directory']=i.count('/')
            d['qty_questionmark_directory']=i.count('?')
            d['qty_equal_directory']=i.count('=')
            d['qty_at_directory']=i.count('@')
            d['qty_and_directory']=i.count('&')
            d['qty_exclamation_directory']=i.count('!')
            d['qty_space_directory']=i.count(' ')
            d['qty_tilde_directory']=i.count('~')
            d['qty_comma_directory']=i.count(',')
            d['qty_plus_directory']=i.count('+')
            d['qty_asterik_directory']=i.count('*')
            d['qty_hashtag_directory']=i.count('#')
            d['qty_dollar_directory']=i.count('$')
            d['qty_percent_directory']=i.count('%')
            d['directory_length']=len(i)
            l.append(d)
        df=pd.DataFrame(l)
        return df
```

- This following code classifies the URLs from the webpage and converts the url feature data into a dataset.

```
m="clickjacking.html"
L=web(m)
D,T,P,Q,F=lis(L)
df1=url(L)
df2=domain(D)
df3= Directory(P)
df4=param(Q)
df5=file(F)
df6=url_attr(L,T,Q)
df7=pd.concat([df1,df2,df3,df4,df5,df6],axis=1)
df7
```

Figure 12: URL features are made into dataset.

- Following code classifies the iframe as a malicious or legitimate one.

```
p={}
pimg=loadImages("predimg/")
data = np.array(pimg)
data = data.reshape([-1,50,50,1])
X_test = data/255
model= load_model('model.h5')
y_pred = np.argmax(model.predict(X_test),axis=1)
p['url']=L
p['phishing']=y_pred
df11=pd.DataFrame(p)
df11
```

Figure 13: Malicious iframe prediction.

- The malicious iframe's border is highlighted with the HTML property.

```
def sep(df11):
    J=''
    for i,j in zip(df11["url"],df11['phishing']):
        if j==1:
            k='<iframe src='+str(i)+' style="border: 5px solid red;" <></iframe> '
            J=J+k
        else:
            k='<iframe src='+str(i)+'<></iframe> '
            J=J+k
    return J
```

Figure 14: HTML property to highlight the malicious iframes.