National College of Ireland

A Standalone tool for Realtime phishing detection

MSc Research Project
Cybersecurity

Shyam Natarajan
Student ID: x23140321

School of Computing
National College of Ireland

Supervisor: Liam Mcabe

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shyam Natarajan		
Student ID:	X23140321		
Programme:	MSc in Cybersecurity	**Year:**	2024
Module:	MSc research Practicum-II		
Supervisor:	Liam Mcabe		
Submission Due Date:	12/12/2024		
Project Title:	A Standalone tool for Realtime Phishing detection		
Word Count:	**7282**	**Page Count 21**.	

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: Shyam Natarajan

Date: 12/12/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

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

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

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

Table of Contents

1 Introduction 3
2 Related Work 4
2.1 Key takes from the literature 7
3 Problem statement. 7
4 Research Methodology 8
4.1 Web application setup 9
4.1.1 System Overview 9
4.1.2 Flask application components. 10
4.2 Email fetching and background scanning. 11
4.3 Email fetching and phishing alerts detection 12
4.3.1 Overview: 12
4.3.2 Imap Email integration. 13
4.3.3 Fetching and processing emails. 13
5 Machine learning based phishing detection. 15
5.1 Overview. 15
5.2 Preprocessing of email content. 15
5.3 Random Forest Classifier training. 16
6 Evaluation 17
6.1 Realtime Evaluation using IMAP email data (phishing_detection.py, app.py) 17
6.2 Kaggle Dataset / Case Study 2 (“ifdof.py”) 18
6.2.1 Calculation metrics and result visualising. 18
6.3 Discussion 20
7 Conclusion and Future Work 21
References 22

Table of Figures.

Figure 1 Statista's report on phishing attacks target organisation. 4
Figure 2 Key take away point from the literature review. 7
Figure 3 List of python scripts used in the project....9
Figure 4 System Architecture 10
Figure 5 Manual triggering script in app.py script. 11
Figure 6 Functional Flow of web application Components....11
Figure 7 Scheduling script for auto triggering phishing detection in app.py....12
Figure 8 Feature description. 12
Figure 9 Email fetching and phishing detection workflow....13
Figure 10 Setting up application password. 13
Figure 11 environmental file with Gmail credentials. 13
Figure 12 Email fetching script to acquire unread email in the email_fetcher.py script. 14
Figure 13 email detection script to classify emails after parsing and preprocessing. 14
Figure 14 Email fetching key functions....14
Figure 15 Random Forest Workflow 15
Figure 16 Model's preprocessing and sample training data. 16
Figure 17 Evaluation metrics used summary. 17
Figure 18 Overview of model's performance. 17
Figure 19 Email testing from different accounts. 18
Figure 20 flask application with phishing detection dashboard....18
Figure 21 Model's evaluation Workflow. 19
Figure 22 Confusion matrix representation 19
Figure 23 Bar chart representation of model's correctness 19
Figure 24 ROC curve representation of model's correctness. 20
Figure 25 Precison-recall curve representation of model's correctness. 20

A Standalone tool for Realtime phishing detection

Shyam Natarajan

X23140321

Abstract

Phishing attacks has evolved with more complicated techniques and has become more difficult to protect the human and organisational resources with right amount of knowledge about them. Protecting the premises asset is one thing also the resources required to achieve that goal is energy draining and takes great toll on the computational resources and human resources of an organisation. To address these issues this project presents a comprehensive phishing detection system that has high performance random forest model script deployed in flask web application to continuously monitor the Inbox of the connected Gmail server. This is a primarily achieved by the Imap library utilised in one of the scripts. To produce absolute results the model is fed with the acquired email data which is TF-IDF vectorised preprocessing. The continuous monitoring of the mail server is achieved by importing one of the modules called "schedule", the scanning is triggered once in 10 mins which maintains no email go unchecked in the inbox. The saved and vectorised random forest module is evaluated with different script to produce visual graphs to make sure the model produces result that good enough to protect the connected mail's inbox. This project involves 2 evaluation phases where first phase of the project's evaluation is done by manual checking with 100s of emails designed to test from separate emails. and the second phase of the model has produced overall accuracy of 85 %. The proposed model achieves primary goal of reducing false positive and false negative outcomes from the model.

1 Introduction

According to Statista's which is out in later part (DEC 9) 2024 report the phishing attacks continues to be a major threat for all types of organisations like social media based, SaaS/Webmail, Financial institution, e-Commerce and logistics based organisation etc (Petrosyan, 2024). with evolved and advanced ways of enumeration. Still people are falling victims to the trick and information are being leaked due to it. The main threat is phishing attack types starts with impersonating to be real emails, websites and communication channels between two different profitable organisations. Which makes it much harder to detect, these threat leads to financial loss, data leakage, and reputation damage among the customer and stake holders.

Email is one of the most common means of communication system between each other. The increase in demand for email communication makes it a higher target for hackers to misuse the communication to medium fulfil their malicious intent. Such email will typically include a malicious URL, suspicious attachments and other social engineering tricks to make fool out of the victim to impersonating as one of the victim's social network people and fulfil their financial gains. Although there are many anti phishing tools available out there. Many of them have some limitations. They rely on external servers, some have delayed detection and some have high false positive rate, making them hard to use.

To address these issues many tools like check point solutions are being available at the market right now especially for the organisation mentioned earlier in this section (Point, 2023). Real time standalone phishing detection and prevention system. IT would need to evaluate email content, embedded URLs and attachments and do so efficiently enough that user is never forced into dependence and their privacy will be kept whole. Such a tool can catch any phishing attempt before they cause any damages to the organisation by any means. This paper proposes such a tool and its comprehensive approach to address this critical issue. Leveraging feature extraction and advanced machine learning model (Random Forest) for efficient operation and reliable outcome.

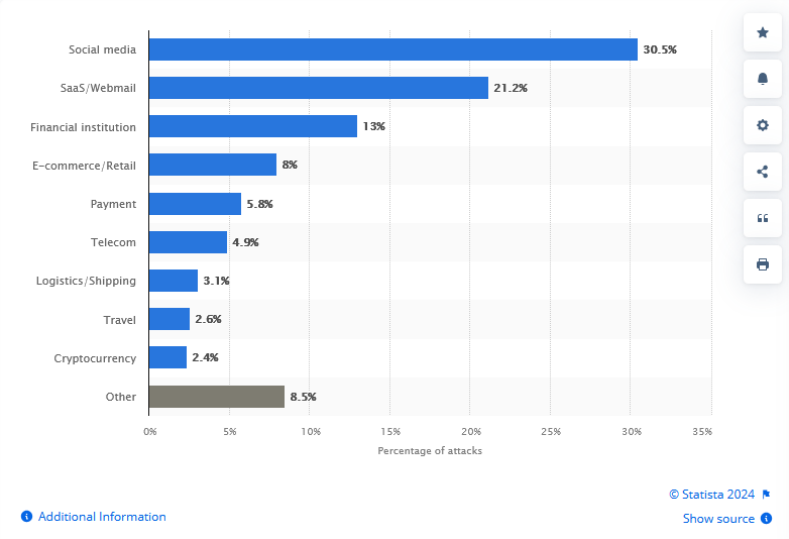


Figure 1 Statista's report on phishing attacks target organisation.

2 Related Work

(Asiri et al., 2024) created the phishing realtime detection system, the authors proposed this as a novel detection framework and proposed this research project for realtime deployment with compelling new features including BiLSTM architecture integrated in a browser extension and a docker container. The framework proposed by the authors is supposed to detect three kinds of phishing techniques including Tiny URLs, Browser in the browser attacks and classic phishing attempts. The authors claimed that the model produced a compelling 99% of precision, recall and f1 score in classifying the URLs contained HTML and java script patterns. Despite all these novelties and compelling results the model has problems in dealing with webpages that contain too much URLs, resulting due to higher latency and degraded performance from the model. The biased and lack of up to date data set is said to be the negative factor that brought down the reason for real time deployment of the tool. The authors haven't tested the tool a by integrating the tool in their browser for realtime evaluation testing that gives less of the reason to taking the authors effort to compel the idea for realtime deployment.

(Gupta et al., 2024) proposed a research paper on phishing email detection framework for enterprise systems. The proposed model is a combination of BERT (Biderictional Encoder Representation from transformers) for feature extraction and a CNN (Convolutional Neural Network) for classification. By leveraging both models BERT and CNN's advantages over the classification and extraction area the authors managed to produce high accuracy of 97.5%. This hybrid approach from the authors illustrates that the research project contains robust architecture for dealing with all kinds phishing attempts. However the authors failed to test the model against varied pattern of the emails and with diverse phishing datasets. The BERT model

contains the scalability issue to deploy it in the realtime system. Due to this shortcoming the proposed model could not be tested or deployed in the realtime mail severs. The authors could have worked on a better choice of model selection that is suitable for scalability and real time deployments.

(Al-Subaiey et al., 2024) proposed a research project on high performance machine learning model to detect phishing attempts via emails. The proposed framework by the authors managed to achieved f1 score of 0.99 % from which is tested with largest publicly available data set that contains 82,455 emails which is impressive but the results is purely based on single dataset outcome when it comes to testing with multiple data set the outcome could have varied on every metric aspects. The author's model is constructed with SVM (support vector machine) with TF-IDF preprocessing. The model utilised large data set which consists of 82,500 emails to hone its capabilities in classifying benign and phishing emails. However the authors made minimal efforts just to rely on a public dataset which undermines the idea of realtime deployment of the model. When tried with realtime deployment the author faced issue led to the deployment of the model to be carried out in future works. The author could have worked on testing the model with data set which contains obfuscated URLs to train the model to obtains absolute results out of it when it deployed in realtime.

(Shafin, 2024) proposed a research project under the name of innovative phishing detection framework which integrates explainable AI techniques (XAI) to enhance the feature selection for the machine learning model used in the phishing detection framework. Again this author tested the hybrid model built with XGBoost and Random Forest with public phishing data set. The Random Forest model came out top producing 97.41% result. The author claimed that the model is performing 0.38ms/ sample and urges the real world deployment with strong results. However the results are heavily relied on the single data set run results. From the results of it the neural networks (KNN) performed poorly with bad accuracies across all the evaluation metrics. For me, the author could have tested with multiple neural networks and ML models with single data set the author could have instead tested the model's performances by integrating with realtime Gmail server and gained the insight on model's performances.

(Tan et al., 2023) worked on hybrid identity based phishing detection techniques that integrates the webpage's visual and textual identity and scans whether it is phishing website or not. This proposal aimed to overcome the shortcomings which is high false positive rate of the conventional identity based detection methods. The authors proposed model consists of hybrid architecture which consists of visual identity discovery model and textual identity discovery model. The models aimed to evaluate the webpages based on the input fields logos, textual content and brand textual elements. The model utilised Webdriver for content manipulation and Open CV for image processing. This model was tested against 2 data sets DS-1 (high ranked websites) and DS-2 (Low ranked websites). The hybrid model produced eye catching results of high accuracy (98.6%) and true positive rate of 99.4% illustrates robust detection capabilities. However the model performed poorly against low ranked websites with true negative rating of 97.8%. In my opinion this model will struggle in the scalability department due to its computational requirements and stress on computational resources. The author can optimise model to perform faster to produce optimal results and to consider the models real time deployment.

(Al Tawil et al., 2024) worked on comprehensive analysis on performances on different ML algorithms such as Random Forest, Logical regression, Decision tree and Multilayer perceptron and used TF-IDF vectorisation for encoding. Among the model tested BERT performed well with detection metrics. Other models fall behind slightly behind the other models. The author provided comprehensive analysis on the available models which can be used for the classification tasks in determining the mail types. The project solely relied on single data set which limits the chances of detecting diverse types of patterns of phishing

attempts. Such limited model cannot be in the conversation for scalability and realtime deployment.

(Brindha et al., 2022) proposed a solution with cuckoo sandbox with deep learning based algorithms for detect email detection and classification which is also known by ICSOA-DLPEC model. The hybrid approach from authors combines email cleaning, tokenisation and stop word elimination and n-gram method for feature extraction. The authors proposed model achieved higher accuracy of 99.72% when tested with Enron data set which contains 7781 email out of which 999 emails were phishing emails. The model demonstrates low training stress on computation and low validation loss which will ensure reliable classification results. However the Enron is a larger dataset this does not represent varied patterns available out there that a hacker can use for phishing attempts. The author could have tested the model's performance with a combination of multiple data set to and observe its performance and computing time to acquire reliable results for realtime implementation.

(Zieni et al., 2023) proposed a research review on phishing detection techniques and algorithms and grouped them into categories based on similarity amongst them. The analysis on the tools stated the strengths, weakness and research gaps in every category. The author supports the idea for integrating ML based models and its capability of handling zero hour attacks and its scalability by adapting itself for the evolving phishing techniques. However, these machine learning techniques can only provide runtime detection and adaptiveness but again depend on the quality of extracted features and training datasets. Features based on URLs, though fast to extract, tend to be vulnerable to evasion techniques such as URL shortening. There will be limited in availability of diverse in phishing patterns when tested with balanced data sets which reduces reproducibility and generalisability. This paper from the authors has reviewed multiple methods of phishing detection by critical analysis. Future works can be done with including diverse dataset which supports deployment and interoperability within the organisations.

(Faris & Yazid, 2021) proposed a rule based application for phishing detection that focuses on interaction between URL and HTML to verify the legitimacy of a webpage. The negative factor for todays machine learning methods is that we need to incorporate different tools for each aspects such as URL, image webpage scanning. The model which is a combination of SVM, Decision tree and naïve bayes, yields 86.6 % accuracy in most cases with different scenarios without incorporating any tweaking in the actual algorithms. The study does not dwell deep into the shortcomings of the model and how to improve the missing 13.4 % of accuracy in rule based detection mechanism, The authors study illustrates the rule based method for handling phishing detection technique, However the model has limitation in adaptability with future novel tactics like hybrid algorithms which naturally yields significantly better results than a unlanguaged algorithm.

(Tanimu & Shiaeles, 2022) proposed a new model phishing detection methodology which is based on the image visualisation of website code and extracting features from various malicious URLs. The authors claims that the existing methods that have a huge gap in their approach so that it can establish a better detection rate and efficiency rate. The authors chose the PhishTank dataset and then converts the website feature into visual representations for classification phases of the model training. The extracted data was converted into images to train the model to obtain effective results. The use of visual image to train the model was a innovative attempt taken by the authors is inspiring for undertaking my project idea and decided to study deep into this paper. However the model has some limitation like the model would struggle in the zero day phishing attempts and it consumes huge resources while training and running the model due to visual image processing.

(Uplenchwar et al., 2022) Proposed an innovative project called phishing attack detection system for text messages. The project used machine learning algorithms techniques such as

Support vector machine, naïve bayes and Random Forest for identifying phishing attempts in text messages. In the end the authors compared the model's performance against each other in identification of phishing attempts. After the model went after each other the Random Forest classification outperformed every other models by significant margin. Every models were trained with blacklists 643,457 URLs to enhance the model's detection capability. From this research paper the performance of the random forest model it inspired my choice of model in this current project for deploying realtime application to give reliable outcome form the Random Forest algorithm. The authors research purely relied on Blacklists dataset which is predefined data set, so it has adaptability shortcomings in the authors approach.

(Dawabsheh et al., 2022) proposed an phishing detection tool based on deep learning algorithms to filter out URLs from the email system. The author developed the algorithms using python programming and the user interface for the tool is developed using the PYQT5. Blacklisted URLs were integrated built in to improve the efficiency and performance of the model in detection department. The tool was trained with a data set of 16,000 phishing sites by utilising 12 methods to bench mark the tool to achieve most accuracy out of the tool. Heavily relying on APIs could drastically reduce the system's performance and raises question in the scalability area for this tool. Author can attempt on remove API reliability of the models to handle its operation to improve the models accuracy and performance to compel the idea to deploy the model for mainstream users.

2.1 Key takes from the literature

After studying multiple literature work similar to the previous works mentioned in the literature review part of this report, there are some key takeaway points worth noting that helped me greatly during the construction of the project.

Pros	Cons
From the surveyed literatures it is evident that random forest classifier model has edge over all the other classification models and it is suitable for the email classification and detection models.	The surveyed literature works always depended on the outcome of single dataset file. Which does not prove much rather than classifying blacklisted email data.
The literature helped me to design the random forest model and gave ideas on how to split the training and testing data.	Most of the authors doesn't deploy or made enough effort to deploy and test their model with realtime mail server which is important part when it comes to testing security tool for its primary purpose.
Some of the literature often contains the advanced patterns how phishing techniques starts and how they work.	Some of the literature often involves deep learning models which produce rather unreliable results which often raises questions on the outcome of the outcome of the model.
Gave insights on how to test and train the model and how to preprocess and make raw data suitable for models training for absolute result.	Most of the literature focused mostly on comparing the model's performance rather than focusing on model's realtime deployment.

Figure 2 Key take away point from the literature review.

3 Problem statement.

Phishing attack are being evolved over the time being and becoming much of energy draining tasks among the cyber security professionals to identify and train the non technical employees in how to handle the phishing attempts. According to a cybersecurity blog from an technological solutions organisation called Visua, in 2024 the traditional methods are now becoming ineffective (Visua, 2024). Some of the main issues are,

- **Heavy reliance on external systems.**

Most of the existing solution mostly deployed on the cloud and external APIs which raises the concern for customers data privacy and protection. API implementation limits the scalability of the system.

- **Lack of Real Time detection.**

Many phishing tools fails to identify some of the sophisticated and advance d phishing attempts, because some of the phishing detection system is not trained to detect advanced patterns. Also most of the system performs poorly when tried to handle thousands of email if any of them contains obfuscated links and attachments some might go undetected.

- **False positive and false negative.**

If the model yields more false positive and false negative rates people and organisation might lose trust on the detection system. If legitimates email get flagged as phishing content people might not even bother to use the detection tool for their work.

- **Scalability and adaptability.**

To address the challenges we need an evolving but also scalable tools that can handle multiple tasks and adaptable to new and advanced phishing techniques with minimal efforts required for training and maintaining.

- **Integrated analysis.**

Most tool out there requires separate modules or separate tool itself to detect different aspects of the phishing email content such as URLs, attachment and obfuscated links. Today all organisation requires a cohesive detection framework to address these solution.

4 Research Methodology

This section of the report provides you a detailed analogy about the methodologies involved in the proposed realtime phishing detection system. This project involves the combination of multiple python scripts to achieve the primary goal of extracting, detecting and classifying emails from realtime Gmail server. This proposed system contains flask web interface which serves user interface with dashboard where the detected emails with respected subjects will be displayed. The Flask application in paired up with machine learning script for phishing detection, email fetching script for extracting email form the Gmail server and performance evaluation script to conduct the stress test on the machine learning model to evaluate its efficiency when it is tested against a cluster of dataset contains more than 19000 unfiltered emails. These scripts were integrated into a single efficient way to identify phishing emails out of the gmail server and process them using Random Forest classifier using lesser computational toll on the system. The scripts involved in the project serves different purpose, The list of scripts involved in the project were,

File name	Purpose

App.py (Flask web application setup)	This python flask script provides user interface with interactive feature to manually trigger the process and also contains the schedule function which automatically trigger the fetching and detection function for every 10 minutes which ensure periodic detection.
Email_fetcher.py (email extractor)	This python script provide the core purpose of connecting and fetching emails from Gmail server in a small matter of time. The IMAP server extracts email and pass it down to other scripts for efficient email detection and classification.
Phishing_detection.py (detection and classification script)	This script contains the core components of the whole project, it serves as base for machine learning model and classify the fetched emails using the Email_fetcher.py script. Further details of this script will be explained in the following sections of the report.
Ifdof.py (Evaluation script)	When connected with Imap gmail server the performance cannot be measured parallely using the Phishing_detection.py script so the same model is deployed in this separate script and which is trained and tested with large data set and acquire graphical representation of accuracy and different evaluation metrics, which supports the performance and reliability of the model under different circumstances.

Figure 3 List of python scripts used in the project.

Each script plays a vital role in achieving the answer for the primary research question which is creating a standalone phishing detection and prevention system to mitigate the phishing attempts in realtime and analyse the email contents to produce low false positive rates results.

4.1 Web application setup

4.1.1 System Overview

This phishing detection system works with a backend email scanning entity with a user friendly user interface with no complicated outputs, options and buttons. Which facilitate the user to efficiently fetch, identify and report the phishing attempts without breaking a sweat. This web application offers an interactive web interface for realtime monitoring the email activities and it helps the user to take appropriate activities when any phishing attempts were detected. Users who were unaware of the threats possessed by the phishing attacks can be aware and be updated with their email activities through a single standalone tool. There are some main components within the web application flask script (app.py) that achieves the proposed functionalities which are,

1. **Email Scanning Service:**
   This service scans the unread emails from the connected Imap Gmail server. The operation of scanning email is achieved by connecting with the email_fetcher.py script which uses imap library which is a built in library available in python programming language.

2. **Web Interface:**
   Web interface is built with flask based application to serve the user interface to monitor the realtime alerts and a dashboard with analysed results from the email scanning. My web interface also provides manual trigger button to start the immediate scanning mechanism and provide analysis results whenever required.
3. **Scheduler and background service:**
   The flask application in this project is designed with a schedular function to trigger periodical scanning mechanism to ensure not to miss any unread emails once every 10 minutes.

   The web application is designed a way that that all these 3 key components works together to provide realtime monitoring and interaction capabilities all within the backend processing. The architecture diagram of these functions can be seen in the below diagram.

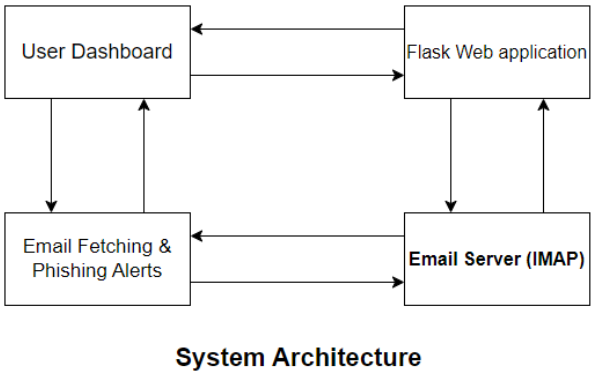


Figure 4 System Architecture

4.1.2 Flask application components.

The proposed phishing detection web application system uses flask to host the web interface, which offers skeletal structure to perform manual or scheduled realtime email scanning and acquiring analysis results on the dashboard. Flask is a lightweight and powerful web application manager makes it an ideal choice for this project's requirements. Some of the components of this flask library utilised in this project were.

- **Manual triggering handle (/Trigger_scan):**
  This function serves as the starting point to trigger the manual scanning incorporated in this project. Once the manual scanning is triggered it gives the user total control of all the modules and scripts like email_fetcher.py, phishing_detection.py, app.py to whenever the user decides to operate at once. This achieves flexibility criteria for any tool to be used by the user.

```python
# Route to manually trigger the email scan via a POST request
@app.route('/trigger_scan', methods=['POST'])
def trigger_scan():
    print("Manual scan triggered.")  # Debugging line
    try:
        mail = connect_to_email(EMAIL_USERNAME, EMAIL_PASSWORD, EMAIL_SERVER)
        if mail:
            fetch_unread_emails(mail)
            mail.logout()
            print("Manual scan completed successfully.")  # Debugging line
        return jsonify({"status": "success", "message": "Email scan completed successfully."})
    except Exception as e:
        print(f"Error during manual scan: {e}")  # Debugging line
        return jsonify({"status": "error", "message": str(e)})
```

Figure 5 Manual triggering script in app.py script.

- **Dashboard endpoint:**
  The endpoint module produce the overview of the scanning processes and phishing alerts produced by the tool during the instant. Which helps the user to identify the potential threats received at the Gmail server end.

The manual and scheduled interaction between the user and the web application can be easily comprehended by looking at the below diagram.

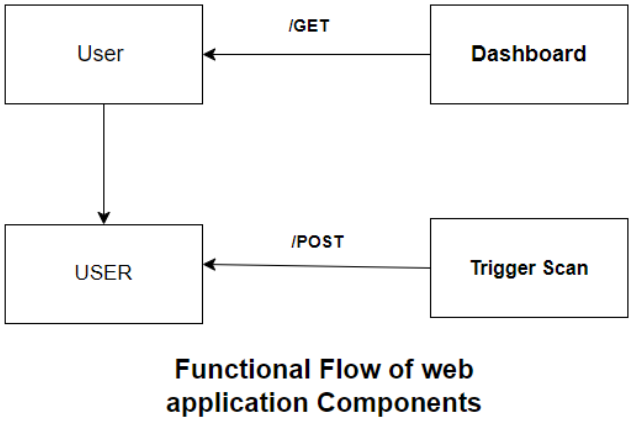


Figure 6 Functional Flow of web application Components

4.2 Email fetching and background scanning.

The designed web application is incorporated with background scanning functionality to achieve consistent email scanning without any intervention between user or Gmail server. This functionality of continuous scanning is achieved by **schedule** library installed in the flask python script. This automates the process of email fetching, scanning and analysing the emails from the imap.Gmail server. In this project I have set the trigger point to 10 mins which triggers the processes once in 10 mins.

- **Schedule email scan:**
  The scheduler installed in the web application is set to 10 mins automated trigger which triggers the scan function used in the email_fetcher.py script which automatically gives you the result of the phishing_detection.py script and ensures the email inbox is monitored continuously for upcoming threats.

```python
# Schedule email scanning every 10 minutes
def scheduled_scan():
    print("Scheduled scan triggered.")  # Debugging line
    try:
        mail = connect_to_email(EMAIL_USERNAME, EMAIL_PASSWORD, EMAIL_SERVER)
        if mail:
            fetch_unread_emails(mail)
            mail.logout()
            print("Scheduled scan completed.")  # Debugging line
    except Exception as e:
        print(f"Error during scheduled scan: {e}")  # Debugging line

schedule.every(10).minutes.do(scheduled_scan)
```

Figure 7 Scheduling script for auto triggering phishing detection in app.py.

- **Background thread for email scheduler:**
  Background thread used in this web application enables the scheduled task could manage multiple operations simultaneously without any background lag without spoiling smooth operations and experience for the user.

Feature	Description
Email scanning frequency	Module used in the flask app.route to set the trigger point to once 10 mins.
User dashboard	Offers a web app interface to show the analysis of phishing alerts.
Manual scan trigger	Trigger the scanning, fetching and detection functionality manually.
IMAP server	Library which connects with the Imap.gmail server.
Multi Threading	Automates background threading used in email scanning.

Figure 8 Feature description.

4.3 Email fetching and phishing alerts detection

4.3.1 Overview:

This section of the function serves as the starting point of the tool to start email connection with the Gmail server and fetching the unread emails automatically and further processing for phishing content detection. This email_fetcher.py function totally responsible for the flawless detection and no emails goes unchecked. The workflow of the script is illustrated in the neat flow diagram below.

Email fetching and Phishing detection workflow

Figure 9 Email fetching and phishing detection workflow

4.3.2 Imap Email integration.

The projects email_fetcher.py script utilises the **imap** library to establish stable connection between the Gmail server and the IDE (vscode) used in this project. Imap ensure the stable and safe connection between the entities using application password which is 16 digit hex hash generated in the Gmail account settings. Further detailed version of this process is given in below screenshot.

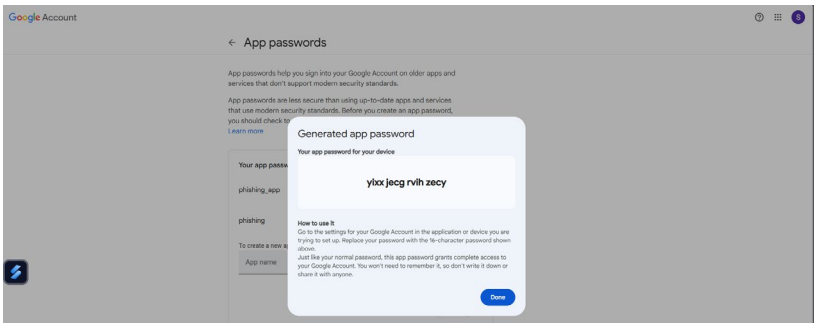


Figure 10 Setting up application password.

The application password serves another usage which is to bypass the multi factor authentication incorporated in the gmail account. So, for the tool's sake we don't have to compromise the accounts basic security needs which ensure the additional layer of security for the Phishing detection tool.

Next phase of the Imap integration involves the credential loading into the email_fetcher.py script. The credentials should be stored as environmental variable to be loaded into the python script. For that **python-dotenv** library should be installed in the script. Environment file loading ensures smooth operation of connecting with the Imap server without any stress delay. Next the imap Gmail function needs to be enabled in the setting section of Gmail application. Which gives access to the IDE to utilise the imap library top connect with Gmail server using the saved credentials in environment variable file.

Once connected the email_fetcher.py script selects and load the unread emails this process will make sure that no unread email go analysed for any malicious content and produce absolute results that a user desires.

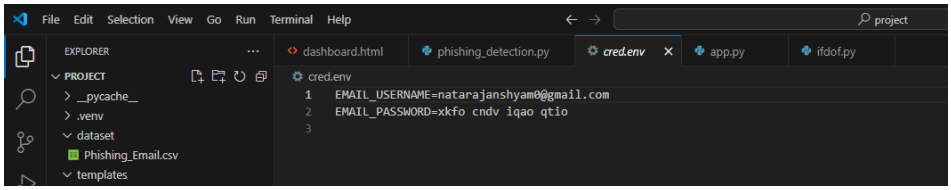


Figure 11 environmental file with Gmail credentials.

4.3.3 Fetching and processing emails.

The email_fetcher.py fetches the unread email form the Gmail server and parse the unread content of the emails to further processing modules incorporated in the email_fetcher.py script. The steps and function involved in the processes were,

- **Fetching unread emails.**
  This function facilitates the script to search and fetch the unread emails in the inbox utilising the imap library commands and parsed into the other parts of the script for further analysis.

```python
# Function to fetch unread emails
def fetch_unread_emails(mail):
    try:
        status, messages = mail.search(None, 'UNSEEN')
        if status == "OK":
            email_ids = messages[0].split()
            print(f"Unread emails found: {len(email_ids)}")
            for email_id in email_ids:
                status, msg_data = mail.fetch(email_id, "(RFC822)")
                for response_part in msg_data:
                    if isinstance(response_part, tuple):
                        msg = email.message_from_bytes(response_part[1])
                        process_email(msg)
        else:
            print("No new unread emails.")
    except Exception as e:
        print(f"Error fetching emails: {e}")
```

Figure 12 Email fetching script to acquire unread email in the email_fetcher.py script.

- **Email Parsing.**
  The whole content profile of each email like subject, body and attachments in the email is extracted and parsed for further analysis. The parsing of content is made possible by converting the HTML content into plain text for efficient translation and detection by the machine learning models used for processing.
- **Phishing detection.**
  A separate scripts I used in this project under the name phishing_detection.py for analysing the email content and search for IOCs (Indicators of compromise). The detection module uses machine learning models for analysing and classifying the emails based on the IOCs identified.

```python
def detect_phishing_email_content(email_text):
    clf, vectorizer = load_model()
    preprocessed_text = preprocess_email(email_text)
    email_vector = vectorizer.transform([preprocessed_text])
    content_flag = clf.predict(email_vector)[0]  # 1 = phishing, 0 = legitimate
    return content_flag
```

Figure 13 email detection script to classify emails after parsing and preprocessing.

Some of the key functions used in the email_fetcher.py script is listed in the below figure,

Function	Purpose
Connect_to_email()	Connects the imap Gmail server with user's system and IDE.
Fetch_unread_email()	Searches and acquires the unread emails form the inbox.
Process_email()	Extracts the email contents like subject, body and attachment files for further translation and processing.
Detect_phishing()	The results from the above 3 function will be analysed using high performance machine learning model used in the phishing_detection.py script.

Figure 14 Email fetching key functions

5 Machine learning based phishing detection.

The random forest classifier model is used in this project because of its robust nature and its extraordinary results from the literature review section on this report in handling large data sets. The model also performs well against overfitting against complex patterns. So choosing random forest classifier model over the other models is felt wise to me and the model stood up to its expectation. The results from this model performance is explained in the later section of the report.

5.1 Overview.

The phishing_detectionn.py script uses TF-IDF vectorisation and Random Forest Classification model to construct a machine learning model for classification tasks used to identify the the phishing attempts made on the email. This component of this project is the vital piece of program to determine the legitimacy of the emails received in the inbox. The workflow of the Random Forest model used in this project is illustrated in the below workflow diagram.

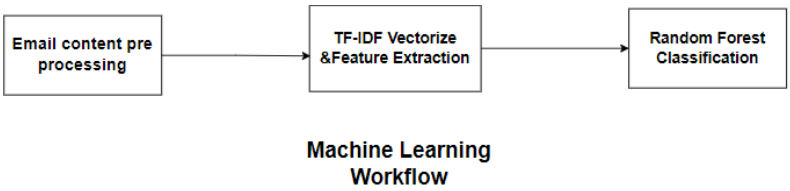


Figure 15 Random Forest Workflow

5.2 Preprocessing of email content.

Preprocessing is a methodology of preparing the raw data into a suitable training and testing data for analysis phase to be fed into the machine learning models. The preprocessing tsrtergies used in the project model is TF-IDF Vectorise and feature extraction to achieve,

- **Lowercasing, Removing URLs and Punctuation.**
  Reducing the noise in the raw data will ensure the consistency in the analysis part and produce accurate results after the training and testing phases of the model. In this project the email data is converted into a lowercase, URL were separated and translated and all other data will be converted into lowercase.
- **Feature extraction using TF-IDF Vectorizer.**
  The cleaned data which is lowercase data will be converted into numerical data which suitable enough to be fed into the training and testing phases of the Random Forest classification model. The TF-IDF based vectorisation will add weights to the words based on the frequency of occurrence in the email and capture the important terms which is training data fed into the models training. From which the model will be able to identify the legitimate email and phishing email. The suspicious email will then get detected and will be added into the phishing alerts dashboard in the flask application.

```python
# Path to save/load trained model and vectorizer
MODEL_PATH = "rf_classifier.pkl"
VECTORIZER_PATH = "tfidf_vectorizer.pkl"

# Preprocess email content for analysis
def preprocess_email(text):
    text = text.lower()
    text = re.sub(r"http\\S+|www\\S+|https\\S+", '', text, flags=re.MULTILINE)
    text = re.sub(r'\S+@\S+', '', text)
    text = re.sub(r'\d+', '', text)
    text = text.translate(str.maketrans('', '', string.punctuation))
    text = re.sub(r'\s+', ' ', text).strip()
    return text

# Sample training data
emails = [
    "Please verify your account at http://phishingsite.com",
    "Your order has shipped! Track it here: http://retailersite.com",
    "Update your banking info at http://securebanking.com",
    "Special offer just for you, click here",
    "Your account has been locked. Verify here",
    "Welcome to our platform! Start shopping now"
```

Figure 16 Model's preprocessing and sample training data.

5.3 Random Forest Classifier training.

Once the data are extracted out of the emails after the preprocessing, then the phishing_detection.py script starts with the training process for the Random Forest classification model. There are few steps involved in the training stages of the Random Forest classifier model they are,

- **Data Split.**
  The data set is split into training and testing set in 80%:20% ratio which yielded better results than the other splits. This split is well suitable for generalising the model to new data sets.
- **Training.**
  From studying multiple literature work on the classification work mentioned in the literature review part of the document I chose to use random forest classifier model for the project. Random Forest is a robust classification algorithm which uses tree like structure to create multiple decision tree during the training and averages the total outcomes of the prediction and will give the final averages of all the predictions.
- **Saving the model.**
  After the training part of the model the trained model and pre processed vectoriser will be saved for future decision makings in realtime phishing detection. Which supports the model greatly in making decision quickly without taking tolls on the computational resource and performances of the model.

Some of the evaluation metrics used to support the correctness of the model is in the table below,

Metric	Description
Accuracy	Total correctness of the model.
Precision	Proportion of true positive over predicted positive.
Recall	Proportion of True positives over actual positives.
F1-Score	Harmonic mean of precision and recall produced by the model.

Figure 17 Evaluation metrics used summary.

6 Evaluation

The evaluation for this project assesses the model's performances using different factors and under different circumstance. This ensures the smooth performance and efficiency of the model in detection and analysis of the email detection. There are two separate phases of evaluation used in this project. First phase of model's evaluation uses realtime email data which is vectorised and processed to obtain the phishing detection results. The second phase of the model's evaluation is incorporated because, the model's performance cannot be determined while handling with realtime email data so I created a separate python script with exact same model to visualise the models efficiency when given with separate public data set of 19,000 emails.

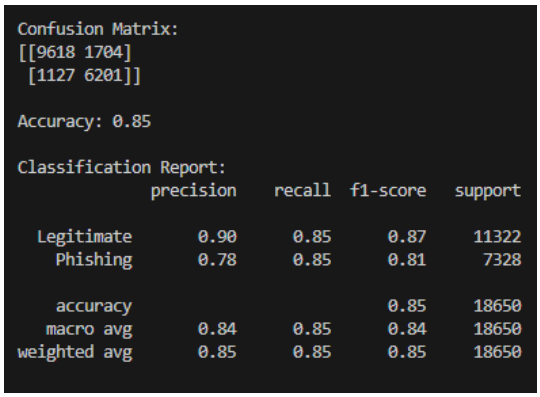


Figure 18 Overview of model's performance.

6.1 Realtime Evaluation using IMAP email data (phishing_detection.py, app.py).

The first part of the evaluation is evaluated using the realtime testing and detection using the Imap email data fetched using the email_fetcher.py script. The acquired data is then subjected to TF-IDF vectorisation which converts the data suitable for training and testing of the random forest classifier model. The training and testing split is set to 80:20 which gives accurate result in most of the cases. The model is tested with 1000s emails which is sent from different accounts from different domains (gmail, Hotmail, protonmail) and the result will be added into the flask applications dashboard. Then I made the tool run for more than 20 minutes straight to test its schedule function, the model works absolutely fine and the expected email to be flagged as phishing alert is added on the dashboard without fail. The series of email testing can be seen In the picture below.

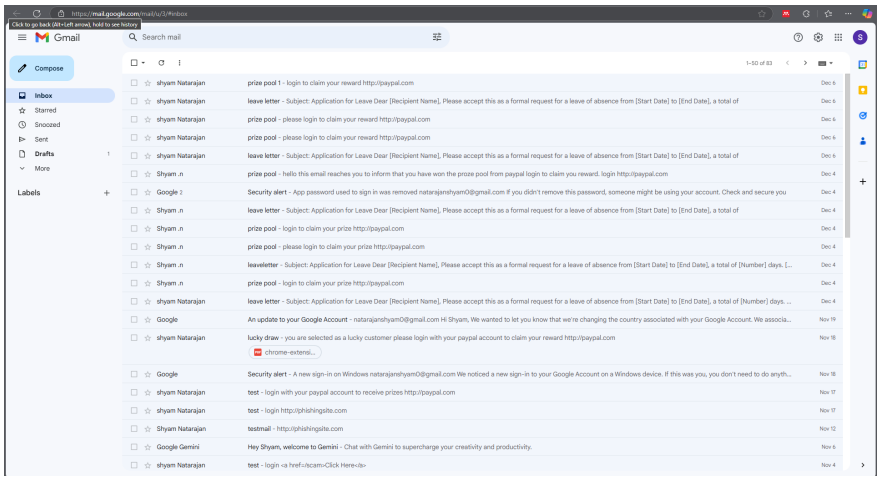


Figure 19 Email testing from different accounts.

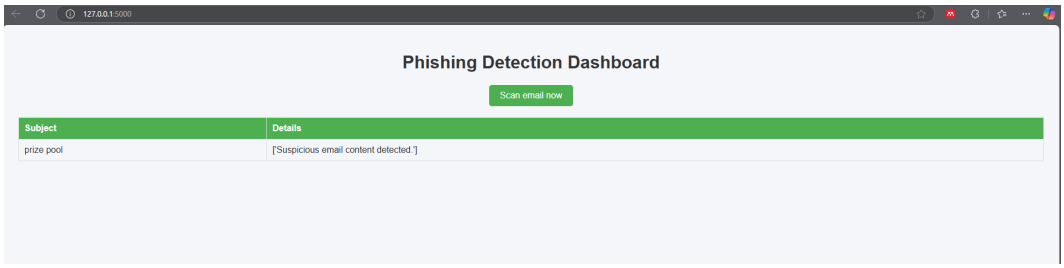
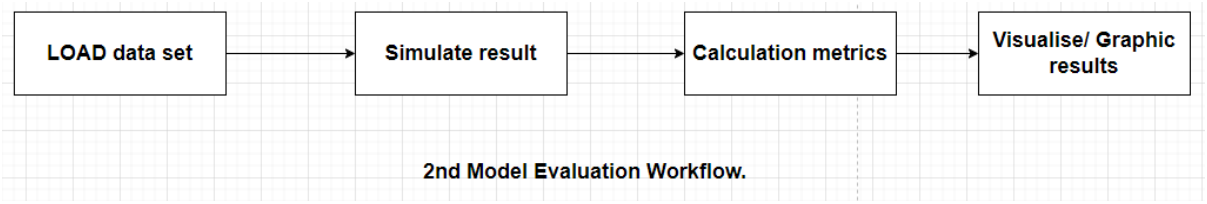


Figure 20 flask application with phishing detection dashboard.

6.2 Kaggle Dataset / Case Study 2 ("ifdof.py")

To evaluate and precisely visualise the results of the model's performance we should use external offline data using different metrics and graphs drawn using scikit learn module in python. We cannot do both the visual evaluation and mail detection using the same script so I decide to create a separate script under the name of "ifdof.py". The saved random forest model from the main phishing_detection.py module with the same training and testing split for the data. The model performed exactly what I expected but fell short in some of the area where the model was expected to identify the JavaScript and html embedded tags which is a flagged mail to initiate phishing attack. The dataset used in the 2nd evaluation part is obtained from Kaggle repository which contains 18650 columns of different email with different patterns with updated advanced and modern phishing emails. The patterns includes embedded and obfuscation methods to stress out the performance of the model (Cop, 2024).

6.2.1 Calculation metrics and result visualising.



2nd Model Evaluation Workflow.

Figure 21 Model's evaluation Workflow.

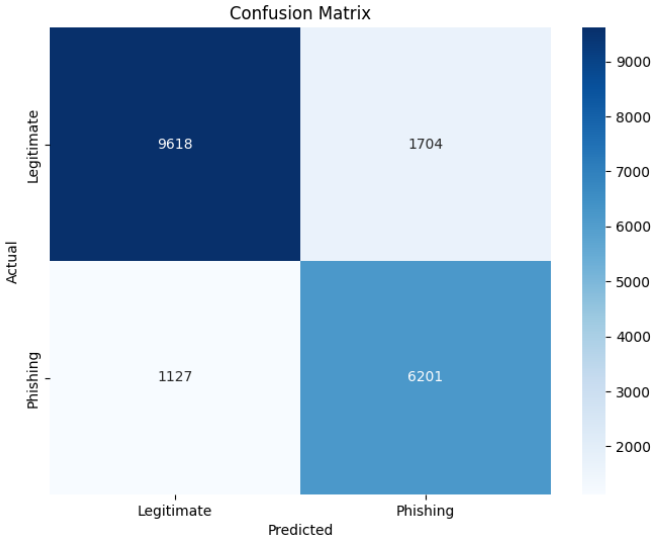


Figure 22 Confusion matrix representation

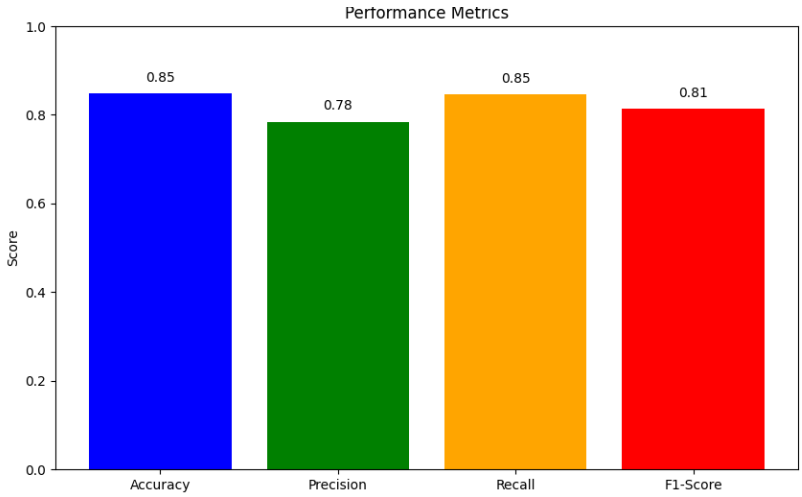


Figure 23 Bar chart representation of model's correctness

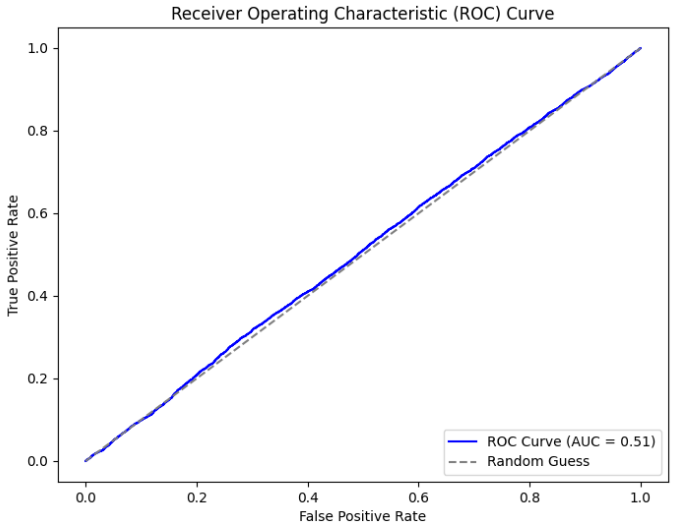


Figure 24 ROC curve representation of model's correctness.

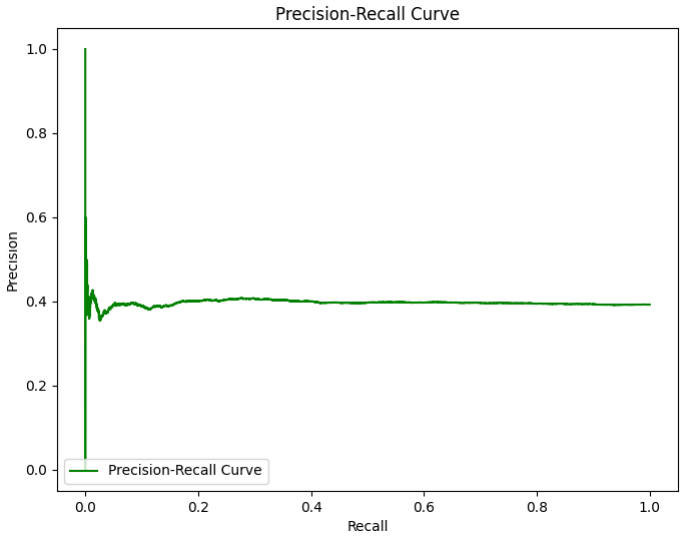


Figure 25 Precision-recall curve representation of model's correctness.

6.3 Discussion

The Realtime evaluation of the model produced just fine result when it was tested against the obvious phishing email contents which is crafted by myself and formulated it as a result of online research from multiple sources. The tool worked well against the obvious phishing emails and successfully classified normal emails as well. However in some cases like embedded java script and HTML tags and obfuscated methods of phishing it fails to identify the email phishing attempts 3 out of 10 times. To exactly support the testing processes I have created separate python script which is exact saved copy of the main detection script which outperformed the manual testing done with realtime email data and produced overall accuracy of 85%. From the above graphs and confusion matrix it is clear that the model has performed strongly with phishing data set from the Kaggle (Cop, 2024). A high of 90 % is achieved in classifying the benign emails and high of 78% accuracy is scored in classifying the phishing emails. The missing score is lost when it went against identifying advanced obfuscated phishing attempts which includes embedding java script and inline html tags. The confusion matrix shows the areas where the model has mis classified the legitimate emails and phishing emails. The ROC curve and AUC curve scores explains how the model has performed in tradeoffs between sensitivity and specificity. From overall part of this report and evaluation results of these two phases, we can strongly see that this proposed model has clear potential in real time deployment and due to its user friendly nature.

7 Conclusion and Future Work

The proposed Realtime phishing detection module is integrated with multiple scripts that monitor the mail service inbox continuously and provide Realtime threat protection against all almost all type of the phishing attempts. The flask application which contains the phishing detection dashboard and manual scan triggering module which allows user to acquire the security information of their respective email account whenever required them. From the evaluation parts of the project the model performed up to my expectation and provided overall 85 % of accuracy across all the testing and made almost perfect predictions.

Although some of the realtime testing which involved embedded and obfuscated code in both the evaluation phase is not detected by the proposed standalone tool so in the future scope of the project I will try to improve the prediction result to near perfect to produce absolute result. Integration of hybrid model will always yield better result than a single model. So in the near future I might try to implement hybrid model of Random Forest and SVM (support vector machine) model to produce better classification and prediction output out of the tool. And as part of the further improvement, and I believe the proposed model can be configured to handle multiple emails and scanning multiple email with separate dashboard for each email can be implemented. Which give compelling reason to deploy the tool as mainstream application for the organisation and sectors demanding the detection tool in their premises.

References

Al Tawil, A., Almazaydeh, L., Qawasmeh, D., Qawasmeh, B., Alshinwan, M., & Elleithy, K. (2024). Comparative Analysis of Machine Learning Algorithms for Email Phishing Detection Using TF-IDF, Word2Vec, and BERT. *Computers, Materials and Continua*, *81*(2), 3395–3412. https://www.sciencedirect.com/org/science/article/pii/S1546221824008117

Al-Subaiey, A., Al-Thani, M., Abdullah Alam, N., Antora, K. F., Khandakar, A., & Uz Zaman, S. A. (2024). Novel interpretable and robust web-based AI platform for phishing email detection. *Computers and Electrical Engineering*, *120*, 109625. https://www.sciencedirect.com/science/article/pii/S0045790624005524

Asiri, S., Xiao, Y., Alzahrani, S., & Li, T. (2024). PhishingRTDS: A real-time detection system for phishing attacks using a Deep Learning model. *Computers & Security*, *141*, 103843. https://www.sciencedirect.com/science/article/pii/S0167404824001445

Brindha, R., Nandagopal, S., Azath, H., Sathana, V., Joshi, G. P., & Kim, S. W. (2022). Intelligent Deep Learning Based Cybersecurity Phishing Email Detection and Classification. *Computers, Materials and Continua*, *74*(3), 5901–5914. https://www.sciencedirect.com/org/science/article/pii/S154622182200354X

Cop, C. (2024). *Phishing Email Detection*. https://www.kaggle.com/datasets/subhajournal/phishingemails?resource=download

Dawabsheh, A., Jazzar, M., Eleyan, A., Bejaoui, T., & Popoola, S. (2022). An Enhanced Phishing Detection Tool Using Deep Learning From URL. *2022 International Conference on Smart Applications, Communications and Networking, SmartNets 2022*. https://ieeexplore.ieee.org/document/9993984

Faris, H., & Yazid, S. (2021). Phishing Web Page Detection Methods: URL and HTML Features Detection. *IoTaIS 2020 - Proceedings: 2020 IEEE International Conference on Internet of Things and Intelligence Systems*, 167–171. https://ieeexplore.ieee.org/document/9359694

Gupta, B. B., Gaurav, A., Arya, V., Attar, R. W., Bansal, S., Alhomoud, A., & Chui, K. T. (2024). Advanced BERT and CNN-Based Computational Model for Phishing Detection in Enterprise Systems. *CMES - Computer Modeling in Engineering and Sciences*, *141*(3), 2165–2183. https://www.sciencedirect.com/org/science/article/pii/S1526149224003163

Petrosyan, A. (2024, December 9). *Global most targeted industries phishing 2024 | Statista*. https://www.statista.com/statistics/266161/websites-most-affected-by-phishing/

Point, C. (2023). *Phishing Detection Techniques - Check Point Software*. https://www.checkpoint.com/cyber-hub/threat-prevention/what-is-phishing/phishing-detection-techniques/

Shafin, S. S. (2024). An Explainable Feature Selection Framework for Web Phishing Detection with Machine Learning. *Data Science and Management*. https://www.sciencedirect.com/science/article/pii/S2666764924000419

Tan, C. C. L., Chiew, K. L., Yong, K. S. C., Sebastian, Y., Than, J. C. M., & Tiong, W. K. (2023). Hybrid phishing detection using joint visual and textual identity. *Expert Systems with Applications*, *220*, 119723. https://www.sciencedirect.com/science/article/pii/S0957417423002245

Tanimu, J., & Shiaeles, S. (2022). Phishing Detection Using Machine Learning Algorithm. *Proceedings of the 2022 IEEE International Conference on Cyber Security and Resilience, CSR 2022*, 317–322. https://ieeexplore.ieee.org/document/9850316

Uplenchwar, S., Sawant, V., Surve, P., Deshpande, S., & Kelkar, S. (2022). Phishing Attack Detection on Text Messages Using Machine Learning Techniques. *2022 IEEE Pune*

Section International Conference, PuneCon 2022. https://ieeexplore.ieee.org/document/10014876

Visua. (2024). *A close look at common challenges in anti-phishing - VISUA*. https://visua.com/a-close-look-at-common-challenges-in-anti-phishing

Zieni, R., Massari, L., & Calzarossa, M. C. (2023). Phishing or Not Phishing? A Survey on the Detection of Phishing Websites. *IEEE Access*, *11*, 18499–18519. https://ieeexplore.ieee.org/document/10049452