

# CLASSIFICATION OF EMAIL HEADERS USING RANDOM FOREST ALGORITHM TO DETECT EMAIL SPOOFING

MSc Academic Internship  
CYBERSECURITY

**OLUWASEUN ODUNIBOSI**

Student ID: X18123970

School of Computing  
National College of Ireland

Supervisor: Mr VIKAS SAHNI

**National College of Ireland**  
**MSc Project Submission Sheet**  
**School of Computing**



**Student Name:** OLUWASEUN ODUNIBOSI  
**Student ID:** X18123970  
**Programme:** MSC CYBERSECURITY **Year:** 2019  
**Module:** ACADEMIC INTERNSHIP  
**Supervisor:** VIKAS SAHNI  
**Submission Due Date:** 12/12/2019  
**Project Title:** CLASSIFICATION OF EMAIL HEADERS USING RANDOM FOREST ALGORITHM TO DETECT EMAIL SPOOFING  
**Word Count:** 5429 **Page Count:** 17

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:** .....

**Date:** .....

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

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

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

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# CLASSIFICATION OF EMAIL HEADERS USING RANDOM FOREST ALGORITHM TO DETECT EMAIL SPOOFING.

OLUWAEUN ODUNIBOSI  
X18123970

## Abstract

Email has become a tool for communication in and around the world in general. The use of email as medium of communication has increased despite the availability of other means of communication like social media and electronic messages. Email has come to stay and so also the threats which come with the use of emails. With the increase in emails, threats like email spoofing, phishing and spamming are on the rise. Researchers have proposed various method for management of email threats which involves classification and filtering of email to deal with the problem.

The motivation of this research is that it that email header contains very important information which can be used in the detection of email spoofing. This paper successfully extracts email header from user inbox using python script, saves the email header in CSV format and successfully classifies user inbox messages using random forest algorithm to detect spoofed or legitimate mail. It also looks at the performance of the script on overhead of the resources it is executed on.

## 1 Introduction

Email is an essential form of communication in terms of information exchange globally. A typical user receives 20-40 email messages in a day. For business and larger organizations, more email is received and sent out. This has increased the working times in processing these mails. As popularity increases, there has also been an increase in mails consisting of irrelevant, spoofed and phishing mail, hence a good classification method is needed to identify these mails. Sean Micheal Kerner (2019)<sup>1</sup> said email users continue to be one of the easiest marks for cybercrime according to latest cybersecurity research.

Email header is a snippet in an email document which holds information such as recipient, sender, routing information and some authentication protocols.<sup>2</sup> An email header has fields that are mandatory like FROM, TO, DATE while the others are not compulsory and widely used like CC and SUBJECT. With this information at hand, an attacker can spoof email header to trick user into thinking the mail is coming from a legitimate source. The method mostly used in this act is email spoofing. Email Spoofing is said to be the forgery of email header by an attacker tricking the user of the mail into thinking the mail is from another source. A user may be tricked into believing a mail is coming from a legitimate source and click on the link or attachment which may redirect user to another site or download malicious content on user computer system. This can be used for fraudulent ways by the attacker through some other attack like phishing, spear phishing attack and spamming. Therefore, many researches have been carried out in order to detect spoofing in mails using method like

---

<sup>1</sup> <https://www.esecurityplanet.com/threats/email-major-attack-vector-security-research.html>

<sup>2</sup> <https://whatismyipaddress.com/email-header>

volatile memory forensics, creation of Sender Policy Framework (SPF) record for all IP address within the organization domain and authenticating the IP address with SPF record of your organization domain and creating a DKIM key and policy which messages can be signed with<sup>3</sup>. This has to an extent helped in detection of email spoofing, but issues still arise like what if system shutdown in the course of acquisition of email header from the memory or user deletes evidence of spoofed mail and shutdown the system before investigation can occur, and are the techniques for spoofing email adequate to tackle this issue?

This research uses a script written in python language to acquire the email header of received mails from user inbox, save in a CSV file and use random forest algorithm for classification of email spoofing.

Random forest algorithm which is an ensembled algorithm is based on the construction of many decision trees in the process of training data and final decision is made using decision of the majority tree. Random forest algorithm can also be called a supervised learning where you have a lot of data and this data are split into training and test set. It reduces overfitting (i.e. time of training data is short) and accuracy is high hence the use of the algorithm in our research.

## RESEARCH QUESTION

This paper solves the problem that arise from previous research work where user system must be powered on through the entire process of acquisition of memory and classification of spoofed email. The methodology checks the performance of random forest algorithm in the classification of email spoofing extracted from received mails in user Gmail base on accuracy and to further ascertain the accuracy; confusion matrix will be analyzed. It also checks the performance of the python script on overhead of the system resources used.

This paper is organized as follows, Section II guides us through the related works, Section III put us through the methodology, Section IV takes us through Implementation, Section V takes us through the Design Specification, Section VI includes Evaluation, Section VII takes us through Discussion of our findings and Section VIII guides us through Conclusion and future work.

- **From:** contains the email address of the sender.
- **To:** contains email addresses of the recipient.
- **Subject:** contains information about the topic of the message.
- **Date:** contains time and date at which mail was sent.
- **Reply-To:** contains mail address that is used to reply back by the recipient.
- **Message-ID:** an automatically generated field, it uniquely identifies the message.
- **Received:** contains information about mail servers involved in mail transmission which can be used to trace the path of message transmission.

**Figure 1: Shows a typical Email Header content.**

(SOURCE: <https://ieeexplore.ieee.org/document/7435764>)

---

<sup>3</sup> <https://ironscales.com/blog/machine-learning-email-spoofing-ceo-fraud/>

## **2 Related Work**

Some significant work has been done in the past as it relates to email spoofing and email header classification and are discussed below in the subsections below.

### **2.1 MEMORY FORENSICS**

Several researches have been carried out as it pertains to email spoofing attack, recently; memory forensics approach have been used in acquiring evidence from volatile (RAM) memory of user machine during cyber investigation. (Mishra et al., 2012) proposed a method which uses an investigative algorithm to check for spoofed addresses in an email by carrying out analysis on email header features. This research was carried out by using dataset (spoofed and legitimate) email from there lab. Four email header field was taken into consideration for their research and are DMARC, DKIM signature and received SPF fields, the proposed algorithm looked for valid value of the field and any invalid value is categorized as unauthorized mail. The algorithm was able to find address spoofed email. The limitation of this research is that not all network traffic can be captured in the network. Additionally, (Jayan and Diya, 2015) proposed different kinds of methods in the identification of spoofing and this was done by analyzing email header. This work focused on email forensics which comes under network forensics. One method proposed was analysis of date and time stamp inside the email header, the proposed method calculate time taken for mail to be received from the sender and compare it with time taken to receive a mail from sending machine. Any deviation from the norm may denote spoofing. Another method proposed was checking domain names inside the received field of email header, a DNS lookup and reverse DNS lookup was done to detect evidence of email spoofing, received field of email address contains IP address that relates to the domain name and if there is any mismatch in the IP address information relating to the domain name, then the mail is categorize as been spoofed. This technique was seen to provide good solution to identification of email spoofing. Subsequently, (Iyer et al., 2017) proposed a method that extract critical evidence from volatile memory of a machine. This method acquires client volatile memory of client machine on a scheduled bases using a forensic tool called Memoryze Mandiant and captured memory dump is scrutinized to detect spoofing attack. The captured memory is first converted to ASCII using Stringtools and then analyzed for evidence of spoofed mail. Spoofed mail was detected and saved in a log file for cyber investigators use. This method guarantees non repudiation. User cannot deny receiving or replying a spoofed email. There result showed that the scheme took 12 minutes to accurately detect spoofed email while preventing false positive and there was no interruption in the functioning of the computer system used. The main limitation of employing this method proposed is that there is no guarantee that system will not power off in the course of capture of volatile memory. This proposed method also skipped some emails when this method was used to process relatively great number of newly received emails in a single execution.

### **2.2 MACHINE LEARNING ALGORITHM**

Machine learning as evolved in resent time and has been used in the classification of email. (Washha et al., 2012) proposed a powerful email header features by using publicly available datasets and applied several machine learning classifiers on header features extracted, this result of this research was evaluated based on performance of the classifiers used. Random forest, Decision tree, voting feature Interval, Random tree, REPTree, Bayesian Network and

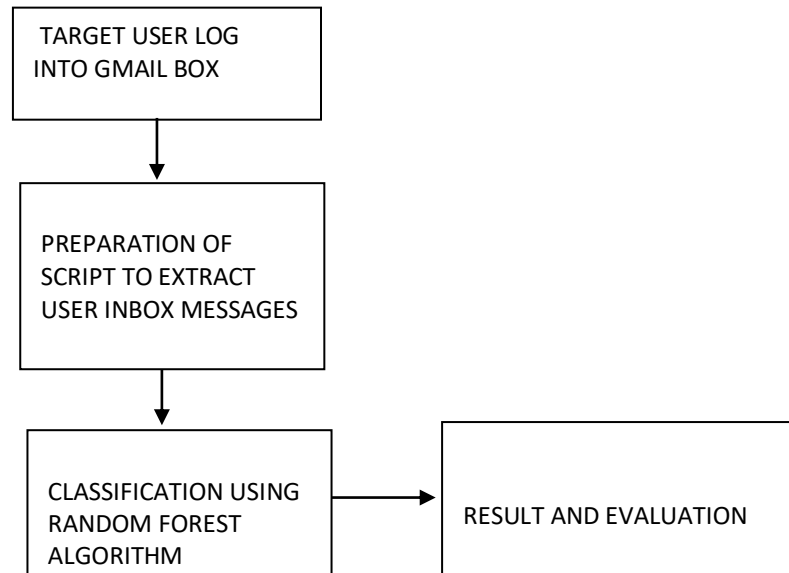
Naïve Bayes was used. The result of the proposed method shows that Random forest classifier performed best with accuracy, precision, recall, F-measure of 99.27%, 99.40%.99.50%. Subsequently, (Mishra and Thakur, 2013) proposed a method for classification of Spam and legitimate mail. The paper used dataset which consist of 9324 records and 500 attributes to train and test the model. The research used three machine leaning algorithm (naïve Bayes, random forest and random tree) on the dataset. The conclusion from the simulation result shows that random forest classifier has more efficiency of 96.389% more than naïve Bayes with 91.664%. Additionally, (Dada and Joseph, 2018) proposed the use of random forest algorithm for classification of email in 2018. Their research was to create a spam filter with better prediction accuracy with a smaller number of features. Dataset from Enron which contains 5180 emails consisting of ham, Spam and normal emails was extracted and random forest algorithm was applied on them using WEKA data mining and simulation, data was normalized and unwanted value or fields was removed before putting data into WEKA, dataset was then split into training set and test set before using the random forest algorithm to classify data. The performance of the random forest algorithm resulted in an accuracy of 99.2%, a false positive of 0.01 which is relatively low, true positive if 0.999 which is high. With the above performance of the algorithm used in their research, it was concluded that random forest can be used in both mail server or mail client to reduce Spam and spoofing. Below shows a table of related works, task performed and how the propose methodology will better the previous works done.

**Table 1: SHOWS RELATED WORK DONE AND PROPOSED WORK**

<b>WORK</b>	<b>TASK</b>	<b>EXTRACTION OF EMAIL HEADER</b>	<b>CLASSIFICATION OF EMAIL</b>
Surekha. Gupta; Emmanuel S. Pilli (2014)	Algorithm to check Email header in DMARC, SPF and DKM for invalid values.	YES	NO
Apeona Jayan; Dija S (2014)	Analysed email header based on date and time stamp and performed DNS lookup	YES	NO
R. Padmavathi Iyer; Manoj Mistra (2017)	Acquired memory dump of system, convert to ASCII and analyzed for evidence	YES	NO
Dada E; Joseph S (2018)	Use of random forest algorithm to create a better classification for email	NO	YES
Proposed Method	Extract Email header from User received mail, store in csv format and use random forest algorithm for classification	YES	YES

### 3 Research Methodology

The methodology this propose research will be using is divided into phases as described below.



**Figure 2: Shows process flow of the proposed methodology.**

#### 1. USER LOGIN WITH EMAIL ADDRESS

A user must login to email address on the target machine of the target machine before extraction of Gmail Inbox. The script for extraction of inbox only works for Gmail and not any other email provider. After login in the user must, user will click on the ‘Manage your Google Account’ on the Gmail interface, select security on the menu by the left and scroll down to Less Secure App Access, this must be turned on for the extraction process to be successful. A user must login in order to find email information after live capture in the volatile memory. The Gmail email provider is selected for this research because it is widely accepted and mostly used according to statistics by Gmail Statistic and Trend (2019)<sup>4</sup>.

#### 2. SCRIPT FOR EXTRACTION OF GMAIL INBOX: the script written in Python language will be used in extraction of Gmail inox messages, Python 3.6 64 bit will be downloaded on the system for this purpose<sup>5</sup>. Python module will have to be imported for this script for work and are **Imaplib** which put into effect a client for communicating with IMAP servers I version 4. The email library which manages the email messages, the **CSV** library which put into effect the classes to read and write data in tabular format. The **datetime** library which uses a strftime () format string. The **argparse** library which will be used to parse command line arguments and the **smtplib** which handles sending email between mail servers.<sup>6</sup>

#### 3. CLASSIFICATION USING RANDOM FOREST ALGORITHM: The random forest algorithm is written in python language and will be run on the PyCharm platform. The PyCharm can be downloaded from<sup>7</sup>, The free open source is downloaded for this

<sup>4</sup> <https://techjury.net/stats-about/gmail-statistics/>

<sup>5</sup> <https://www.python.org/downloads/release/python-360/>

<sup>6</sup> <https://docs.python.org/3/library/>

<sup>7</sup> [https://www.jetbrains.com/pycharm/download/#section=windows.](https://www.jetbrains.com/pycharm/download/#section=windows)

research. For our random forest algorithm written in python language to function well on PyCharm, we must import libraries and the libraries to be use are Pandas, RandomForestClassifier, make\_classification, train\_test\_split, metrics, classification report, confusion matrix and accuracy\_score. Pandas stand for Python Data Analysis library and it will help us to structuring our data, changes our CSV file to a python object rows and column called dataframe. RandomForestClassifier will be use for classification of our dataset. Make\_classification will be used to generate a random n-class classification for our dataset. Train\_test\_split will split our dataset into training test and test set, Metric library is used to evaluate the performance of our random forest algorithm. Classification report library is used to compare our classification model visually and Accuracy\_score will be used to check the accuracy of our result.

4. EVALUATION OF RESULT: The following performance metrics used by Washha (2017) will be use in classifying our algorithm. Below are the equations to use for the performance metrics:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{False Positive} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

$$\text{False Negative} = \frac{\text{FN}}{\text{FN} + \text{TP}}$$

$$\text{F- Measure} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Where:

TP (True Positive): number of Spoofed mails correctly classified.

FP (False Positive): number of Spoofed mails incorrectly classified.

FN (False Negative): number of legitimate mails incorrectly classified.

TN (True Negative): number of legitimate mails that are correctly classified.

Accuracy: Is the fraction of all messages classified by the classifier.

Recall: If the number of actual spoofed mail and actual legitimate mails is correctly classified, what is the percentage of the spoofed, legitimate mail?

Precision: Probability that a legitimate, spoofed mail is misclassified.

## 4 Design Specification

The design of the python script for extraction of email header and classification using random forest algorithm is described below. The full script for the extraction and classification of email header can be found in the configuration manual. Some of the code use in this research follow



general convention of using the module download and therefore no change was done on them. More information on this codes can be found on this site.<sup>8</sup>

Various libraries as explained in the methodology are imported into our code, Figure 3 shows the various libraries used in the program

```
import smtplib
import time
import imaplib
import email
import csv
from datetime import datetime
import random as rf
import argparse
```

**Figure 3: shows various libraries imported for the script.**

The user will need to provide login details of the Gmail in the correct format in the configuration section as shown in Figure 4 below;

```
#GMail login

#Title:How to Read Email From Gmail Using Python, 2017. . Code Handbook.
#Author: JAY
#Date: 2017
#Availability: https://codehandbook.org/how-to-read-email-from-gmail-using-
python/#comment-4217804161

SMTP_SERVER = "imap.gmail.com"
SMTP_PORT   = 993
FROM_EMAIL  = "cainersteph@gmail.com"
FROM_PWD    = "Analogue-1234"
```

**Figure 4: shows login details for Gmail, port and Server**

Line 30-44 specify the filename the extracted email is been sent to, number of emails to send and how error is handled in the course of extracting email from user inbox. Figure 5 shows the line of code that handles that reading of the mail from inbox and error handling.

```
# -----
#
# Read mails from GMAIL Inbox
#
# -----
def read_mails(filename, email_counts):

    #Handle connection exception
    try:
        #connect to smtp server
        mail = imaplib.IMAP4_SSL(SMTP_SERVER)

        #Login
        mail.login(FROM_EMAIL, FROM_PWD)
        mail.select('inbox')
```

**Figure 5: shows Reading of mail and Error Handling**

Line 51-57 shows how the mail will be extracted, it divide the received mail into two arrays namely `first_email_id = int(id_list[0])` and `latest_email_id = int(id_list[-1])` and fetching this mail from the array is by fetching the latest mail first follow by the first mail. The FOR-loop

---

<sup>8</sup> <https://codehandbook.org/how-to-read-email-from-gmail-using-python/#comment-4217804161>

statement means that if number of emails to extract is greater than 0, then read, if not; do not read from the inbox. Figure 6 show the code that handles that

```
#Process emails

#Title:How to Read Email From Gmail Using Python, 2017. . Code Handbook.
#Author: JAY
#Date: 2017
#Availability: https://codehandbook.org/how-to-read-email-from-gmail-using-
python/#comment-4217804161

id_list = mail_ids.split()

first_email_id = int(id_list[0])
latest_email_id = int(id_list[-1])

#Fetch emails from latest to old
for i in range(latest_email_id,first_email_id, -1):
    typ, data = mail.fetch(str(i), '(RFC822)' )

    if (email_counts > 0):
        for response_part in data:
            if isinstance(response_part, tuple):
```

**Figure 6: Processing of mail and Fetching of Mails.**

Line 68 to 81 shows the preparation of our email to output file, as seen in the code, the messages to be stored in the csv file is email subject, email from, email to, message and date field, more fields can be added depending on users need. The content is then print out and saved in the output file. Figure 7 show the line of code that deals with that.

```
#Prepare output to CSV file
email_subject = msg['subject']
email_from = msg['from']
email_to = msg['to']
msg_id = msg['message-id']
date = msg['date']

#Message
csv_content = [email_from, email_to, email_subject, msg_id, date]

print("Reading and storing messages in " + filename)

#Save message to CSV file
save_output(filename, csv_content)
```

**Figure 7: Saving of Output**

Line 23-26 shows the function that define where the mail will be saved and how, the 'a' in quote means append and this do not allow us to erase existing file while writing to the csv file, the writer.writerow(row) will write each mail in separate row so existing row wouldn't be deleted or overwritten. Figure 8 shows the function that write saved mail to csv

```
def save_output(filename, row):
    with open(filename, 'a') as f:
        writer = csv.writer(f)
        writer.writerow(row)
```

**Figure 8: Writing of Mail to CSV**

Some argument was parsed into the python script using the python argument parser argparse, the first function is to read mail from the Gmail inbox online, the second function is to read email

from extracted dump. This is needful because investigator may want to read mail that have been earlier extracted and saved without login in to Gmail account, Line 100-104 shows the argument passed to the code as seen below

```
def main():
    # initiate the parser
    parser = argparse.ArgumentParser()
    parser.add_argument("-e", "--emails", help="Input the number of emails to read")
    parser.add_argument("-d", "--dump", help="Provide an exist email header dump file")
```

**Figure 9: Parser argument in use**

Execution of the argparser above can be seen below, if default email count is not specified, only 1,000 mail in user inbox will be read, this can either be read from already created dump email file or directly from Gmail and result is saved in csv file.

```
# Execute command line arguments
args = parser.parse_args()

if (args.emails):
    email_counts = int(args.emails)
else:
    #Default email counts
    email_counts = 1000

if (args.dump):
    #Read emails for local file
    filename = args.dump
    print("Email header file to classify: {}".format(filename))
else:
    #Read emails from gmail
    #Create csv file
    filename = "csvfile_{}.csv".format(datetime.now().strftime("%H-%M-%S"))

    #CSV Fields
    fields = ['From', 'To', 'Subject', 'Message-ID', 'Date']
    save_output(filename, fields)

    filename = read_mails(filename, email_counts)
```

**Figure 10: Execution of argparse**

The Random forest code is included in the Email\_Spoofing\_Classifier\_final code and it prompt the user to decide if the user want to classify data or not, if yes, the Email\_Spoofing\_Classifier\_final code automatically classify the mail using the random forest algorithm. Figure 11 shows the line of code below

```
confirm = input("Do you want to classify the data as well? (Yes or No) ");
confirm = confirm.lower()

if ("yes".find(confirm) != -1): #classify data if yes
    rf.random_forest_algo(filename)
```

**Figure 11: Classification of mail using randomforest.py**

For the classification using random forest algorithm, the email, Python libraries as explained in the methodology section is imported to help in classification of our extracted email header. Figure 12 below shows the libraries used.

```

import warnings
from sklearn.exceptions import DataConversionWarning
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

```

**Figure 12: Libraries for classification**

Line 14-16 shows the start of the classification algorithm, it also reads the csv file generated by the python script (Email\_Spoofing\_Classifier\_final). This is represented in the code below.

```

def random_forest_algo(filename):
    df = pd.read_csv(filename)
    df.head()

```

**Figure 13: Read CSV file**

Line 18-19 print out the dataframe of the generated CSV file as shown in Figure 14 below

```

with pd.option_context('display.max_rows', 1000):
    print (df)

```

**Figure 14: printout 1,000 mails in rows**

Line 21-28 split the dataset into training and test set, `n_samples=1,000` is the number of samples classified by the random forest algorithm. The `n_features=2` comprises of `n_informative` which is the covariance of our sample and `n_redundant` features of 0 drawn at random. `n_estimators = 20` which specifies the number of trees in the forest. A `test_size = 0.3` which represent the number of datasets to include in the test split, `random_state = 5` is chosen in order to get a fixed accuracy result.

```

X, y = make_classification(n_samples=1000, n_features=2, n_informative=2,
n_redundant=0, random_state=5)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
    X_train
    X_test
classifier = RandomForestClassifier(n_estimators=20, random_state=5)
classifier
classifier.fit(X, y)
print(classifier.feature_importances_)

```

**Figure 15: Code to split data into training and test data.**

Line 34-35 get parameters for the classification and print the accuracy of the classification. Result of the accuracy is seen in the evaluation section of this project.

```

# Accuracy
print(metrics.accuracy_score(y_test, y_prediction))
classifier.apply(X)
classifier.get_params(deep=True)
classifier.predict_log_proba(X)
classifier.predict_proba(X)
classifier.score(X, y)

```

**Figure 16: Calculate Accuracy of the Classifier.**

Line 42- 45 compute the confusion matrix which tells us understand the performance of the random forest classification model. Figure 17 shows the implementation of the confusion matrix.

```
# Confusion Matrix
print(confusion_matrix(y_test,y_prediction))
print(classification_report(y_test,y_prediction))
print(accuracy_score(y_test, y_prediction))
```

**Figure 17: Confusion Matrix Implementation.**

## 5 Implementation

The research is performed on a Windows 10 64bits Operating system, ICORE 5, 500GB with 8GIG RAM. The Gmail application is opened from the browser, login credential is provided. Navigate to Gmail settings. Click on security and scroll to Allow less Secure Apps, once done a mail is sent to the inbox prompting user of a less secure app access to Gmail. Click yes and open the PyCharm application. Navigate to the file where the code is stored. The email\_analyzer.py is opened. Gmail login information is provided on the code itself as seen in FIG. 2. Connection to SMTP server is done and error that may occur during reading of emails from user Gmail is handled using the code shown in Fig.3. Email fetched from Gmail inbox are split into an array (Latest\_email\_id and First\_email\_id). This is done so the latest email can be fetched first followed by the last. The FOR-loop statement in the code means that any mail greater than zero will be fetched, if not, the code stops reading mails as shown in Fig. 4. The output of the code to be written to CSV file are listed and saved in CSV format as shown in Fig. 5. After the extraction and saving of email header to CSV file, User is asked if data should be classified or not. If YES is selected, the code classified the extracted emails saved in CSV file using the randomforest.py code.

For classification using random forest algorithm, Python libraries are imported as explained in the methodology section. The dataframe of the extracted email is printed out in rows and column which gives us a better view of our dataset. Dataset of 1,000 mail is split into training and test set. 30% of the mail is used as test set while the rest is used as training set. Accuracy of the classifier is calculated and to further know how the random forest classifier performed by performing a confusion matrix.

## 6 Evaluation

In order to ascertain the success of this research, performance of extraction process and performance of the random forest algorithm is compared with previous research done by R. Padmavathi Iyer; Manoj Mistra (2017). Section 6.1 and section 6.2 below discuss the evaluation of the methodology used.

### 6.1 Experiment / Case Study 1

Comparing the extraction process of email shows that the methodology used in extraction of user email header from Gmail outperformed the one used by R. Padmavathi Iyer; Manoj Mistra (2017) in which email header are extracted from volatile memory of the system, the propose system was able to extract 1,000 user inbox mail and perform classification in less

than 9:20 (mm:ss) as compared to previous research which was completed in 12.10 (mm:ss). The performance of the script as it relates to the computer system is shows that the Script use 0.77MB of memory compared to previous research which used 3.4 MB of memory, the python script of propose system also use 0.1% of disk space compared to previous research which used 25.7%. Figure 18 below shows a tabular view of our result. This shows that the propose method took lesser time to execute while requiring minimum overhead on system resources.

Process	Memory(MB)	Disk(MB/s)	CPU(%)	Time(mm:ss)
Memoryze	1.9	82.1	6.0	2:30
Strings	0.9	24.7	14.6	7:30
QGREP Utility	0.6	45.6	5.1	00:10

Total time taken (mm:ss) = 12:10.

PROCESS	MEMORY(MB)	DISK(MB/S)	CPU (%)	Time (mm: ss)
SCRIPT	0.77	0.1	Varies	8:50

Total time Taken (mm: ss) = 9:20

**Figure 18: shows performance of previous research and propose research on the system used.**

## 6.2 Experiment / Case Study 2

To further check the performance of our research against previous research done in the area of spoofing, the accuracy of the random forest is check for email spoofing accuracy and from result as seen below shows that Accuracy of previous research by R. Padmavathi Iyer; Manoj Mistra (2017) achieved an accuracy of 0.93% while the accuracy of the propose methodology using python script was 0.99%. Figure 19 show the result from previous research and propose research.

EMAIL SERVICE PROVIDER	NO OF RECEIVED SPOOFED EMAIL DETECTED	ACCURACY
Gmail.com	10	93.4%

```

[1000 rows x 5 columns]
[0.22386201 0.77613799]
0.9966666666666667
[[144 0]
 [ 1 155]]

```

**Figure 19: Accuracy result for previous research and proposed research.**

The result using python script shows the following below;

TP (True Positive): number of Spoofed messages correctly classified as 155.

FP (False Positive): number of legitimate mails are that misclassified as 0.

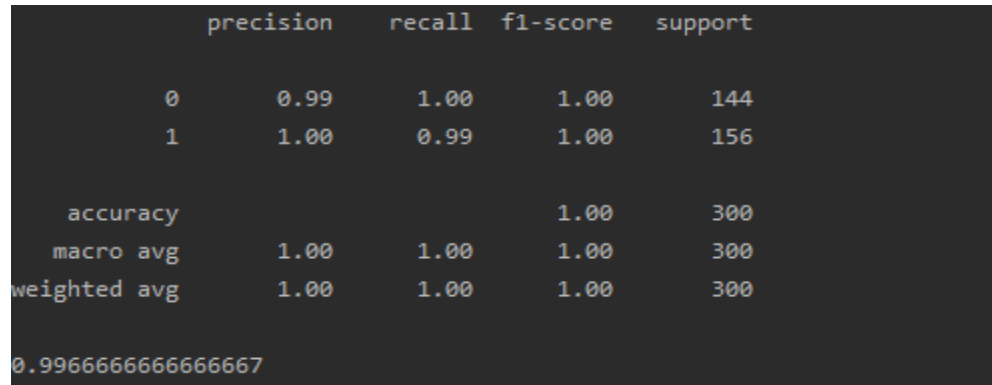
FN (False Negative): number of Spoofed messages that are misclassified as 1.

TN (True Negative): number of legitimate messages that are correctly classified as 144 and

Accuracy: Is the fraction of all messages classified by the classifier of 0.99 (99%).

### 6.3 Experiment / Case Study 3

To further ascertain the accuracy of the result given by above for propose method, we calculate the confusion matrix based on recall and precision. The method using python script produce the following result as shown in Figure 20.



```
precision recall f1-score support
0 0.99 1.00 1.00 144
1 1.00 0.99 1.00 156

accuracy 1.00 300
macro avg 1.00 1.00 1.00 300
weighted avg 1.00 1.00 1.00 300

0.9966666666666667
```

**Figure 20: result of confusion matrix to ascertain accuracy of our propose method**

The result using python script shows the following below;

Recall: performance of recognised Spoofed email of 1.00%.

Precision: Probability that a legitimate mail is misclassified of 0.99.

F1-Score: the weighted average for precision and recall is 1.00%.

Weighted average: sum of all email after multiplying their respective email proportion is 1.00%.

Macro Average: Is the mean of all email classified which is 1.00%.

### 6.4 Discussion

The proposed research was implemented, and it successfully classified spoofed email from extracted email header from user Gmail (received mail). The output of the volatile memory forensics for detecting email spoofing proposed by R. Padmavathi Iyer; Manoj Mistra (2017) proved that when handling memory dump of large size, the algorithm skipped some email. The time taken also to complete overall process in methodology is also relatively high and accuracy for extraction and detection of spoofed mail in Gmail is lower compared to one used in this research. Using the python script; result proved that the number of mails to extract from user Gmail can be specified. The time taken to calculate overall process is low compared to previous research. The python script also achieved an accuracy of 0.99 compared to previous research which took 0.93. The result on overhead of system also shows that the python script didn't have much effect on general working of the system. Though CPU usage varies in the python script, this is due to the python script processing many mails at the same time. This experiment can be performed on a system with higher configuration to see the performance of the python script. The random forest algorithm also shows signs of imbalances in the classification of email (where classes are not represented equally), this can be improved upon by using dataset available in public domain and lastly user approval to use credentials in the extraction of user Gmail.

## 7 Conclusion and Future Work

This methodology was able to check the performance and effectiveness of random forest algorithm in the classification of email spoofing extracted from received mails in user Gmail. It also checks the time of execution, accuracy and performance on overhead of the system resources used. The research executed in 9.20 (mm: ss) and achieved an accuracy of 0.99. It also has little effect on overhead of the resources used in execution. The research solves the problems that arises from keeping the system on during live capture of memory as used in previous research, it eliminates skipping of emails when large size of memory dump is involved.

Future work for this research could be in extending the python script to work with other email provider to check the versatility and performance of the script.

## References

- Dada, E., Joseph, S., 2018. Random Forests Machine Learning Technique for Email Spam Filtering.
- Gupta, S., Pilli, E.S., Mishra, P., Pundir, S., Joshi, R.C., 2014. Forensic analysis of E-mail address spoofing, in: 2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence). Presented at the 2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence), pp. 898–904. <https://doi.org/10.1109/CONFLUENCE.2014.6949302>
- Iyer, R.P., Atrey, P.K., Varshney, G., Misra, M., 2017. Email spoofing detection using volatile memory forensics, in: 2017 IEEE Conference on Communications and Network Security (CNS). Presented at the 2017 IEEE Conference on Communications and Network Security (CNS), pp. 619–625. <https://doi.org/10.1109/CNS.2017.8228692>
- Jayan, A., Dija, S., 2015a. Detection of spoofed mails, in: 2015 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC). Presented at the 2015 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), pp. 1–4. <https://doi.org/10.1109/ICCIC.2015.7435764>
- Mishra, R., Thakur, R.S., 2013. Analysis of Random Forest and Naïve Bayes for Spam Mail using Feature Selection Catagorization. <https://doi.org/10.5120/13844-1670>
- Nizamani, S., Memon, N., Glasdam, M., Nguyen, D.D., 2014. Detection of fraudulent emails by employing advanced feature abundance. Egyptian Informatics Journal 15, 169–174. <https://doi.org/10.1016/j.eij.2014.07.002>
- (PDF) Email Classification Research Trends: Review and Open Issues [WWW Document], n.d. URL [https://www.researchgate.net/publication/316903018\\_Email\\_Classification\\_Research\\_Trends\\_Review\\_and\\_Open\\_Issues](https://www.researchgate.net/publication/316903018_Email_Classification_Research_Trends_Review_and_Open_Issues) (accessed 10.28.19).



Rajput, A.S., Sohal, J.S., Athavale, V., 2019. Email Header Feature Extraction using Adaptive and Collaborative approach for Email Classification 8, 7.

Washha, M., Khater, I., Qaroush, A., 2012. Identifying Spam E-mail Based-on Statistical Header Features and Sender Behavior. Presented at the ACM International Conference Proceeding Series. <https://doi.org/10.1145/2381716.2381863>