Test Utility for Live and Online Testing of an Anti-Phishing Message Security System

by

Dhruvalkumar Patel

B.Eng. Gujarat Technological University, 2014

A Report Submitted in Partial Fulfillment of the Requirements for

the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering

Supervisory Committee

Test Utility for Live and Online Testing of an Anti-Phishing Message Security System

by

Dhruvalkumar Patel

B.Eng., Gujarat Technological University, 2014

Supervisory Committee

Dr. T. Aaron Gulliver (Department of Electrical and Computer Engineering)

Supervisor

Dr. Issa Traore (Department of Electrical and Computer Engineering)

Co-Supervisor

Table of Contents

Supervisory Committee ii
List of Figures iv
List of Tables v
Abstract vi
Chapter 1: Introduction 1
1.1 Phishing Life Cycle 2
1.2 Project Objectives 3
1.3 Approach and Tasks 5
1.4 Report Outline 6
Chapter 2: Background on Phishing 7
2.1 Phishing Attack Vectors 7
2.2 Phishing Motivation 8
2.3 Phishing Detection Techniques 9
Chapter 3: Proposed Test Framework 11
3.1 Data Collection 11
3.2 Create Testing Accounts 14
3.3 Reading and Sending eml Files 15
3.4 Email Segregation Results for Gmail and Yahoo Test Accounts 20
Chapter 4: Partial Test Automation for the Phishing Detector 23
4.1 Module Under Testing 23
4.2 Unit Testing 23
4.3 Testing Framework 24
4.4 Creating a JUnit Test Suite in the Spring Tool Suite 25
4.5 Examples of Test Units with Heuristic Rules 26
4.6 Test Cases and Test Results 32
Chapter 5: Conclusion and Future Work 34
References 35

List of Figures

Figure 1.1 The phishing life cycle……2
Figure 1.2 The ISOT phishing security system……3
Figure 3.1 The Mbox viewer……12
Figure 3.2 The conversion process from mbox folder to eml files……13
Figure 3.3 Example of eml files generated after conversion using the mbx2eml tool……14
Figure 3.4 The Anaconda Navigator Desktop application in Windows 10……16
Figure 3.5 Jupyter Notebook launched on a web browser……16
Figure 3.6 Creating a new project……17
Figure 3.7 New project opens in a new tab……17
Figure 3.8 Flowchart of eml files reading/sending service implementation……18
Figure 3.9 Allow access to less secure apps on Google……19
Figure 3.10 Allow access to less secure apps on Yahoo……20
Figure 4.1 STS after importing the project……25
Figure 4.2 Creating a new test suite class……26
Figure 4.3 Creating the name of the test suite……26
Figure 4.4 Test unit for the Alexa Rank Page rule…… 27
Figure 4.5 The result of the test suite detected as phishing……28
Figure 4.6 The result of the test unit detected as legitimate……29
Figure 4.7 Test unit for the Age of Domain rule……30
Figure 4.8 The result of the test unit detected as legitimate……30
Figure 4.9 Test unit for the certificate validation rule……31
Figure 4.10 The result of the test unit detected as legitimate and the tests are passed……32
Figure 4.11 Some test cases are failed……33
Figure 4.12 All test cases are passed……33

List of Tables

Table 3.1 Email datasets ……12
Table 3.2 SMTP server parameters ……19
Table 3.3 Default filtering results (number of messages) for Yahoo……21
Table 3.4 Default filtering results (number of messages) for Gmail ……21
Table 3.5 Phishing detection performance for Yahoo and Gmail ……22

Abstract

The Internet has become a basic human need as many sectors such as banking, education and e-commerce are highly dependant on the Internet. At the same time, security has become one of the top concerns for Internet users across the globe. One of the top threats faced by most users is phishing scams. Phishing is a nontrivial problem which involves deceptive messages (emails and SMS messages) that trick users into willingly revealing confidential information like passwords, bank account numbers, and social insurance numbers. The goal of this project is to develop a mechanism and set of utilities to test a new phishing detection system. This mechanism involves setting up several test email accounts on Yahoo and Gmail, and populating these accounts from different datasets including legitimate, phishing, and spam emails. This is used for real-world and live testing of the new phishing detection system. Additionally, a collection of unit test cases is created and executed using a test automation framework. This allows for testing and improving the quality of the software implementation of the phishing detector.

Chapter 1: Introduction

One of the top cybersecurity threats currently facing organizations and individuals is phishing attacks. Phishing takes advantage of the credulity of people to convince them to perform actions that may have dire consequences, such as identity theft, ransomware infection, account takeover, and banking theft. At the same time, the detection and prevention of phishing attacks is a big challenge because attackers are working hard to refine their techniques to bypass existing anti-phishing techniques.

In a typical phishing scam, the attacker designs a fake website that looks like an existing legitimate site by copying and making minor changes so that a user is not able to differentiate between the legitimate and fake websites. Most of the time the difference is in the URLs corresponding to the websites.

Another way for phishers to deceive online users is by sending them emails which urge them to perform actions such as clicking on a link or visiting a compromised website. Once the user falls into the trap by clicking the link sent in the email or by logging onto the fake website, the phisher will collect sensitive information (e.g. username/password) from the user that may be used illegally.

A common vehicle used to propagate phishing scams is spam emails. Spam is unsolicited email. Although it could be used to spread a phishing scam, its purpose is much broader. Spam emails are mostly used for commercial advertising and other forms of non-commercial purposes. The difference between generic spam and phishing is that the spam authors do not attempt to steal sensitive information from the recipients. Conversely, phishing attacks are meant to acquire user information such as login IDs and passwords so that the attacker can get access to things such as social network and bank account details.

1.1 Phishing Life Cycle

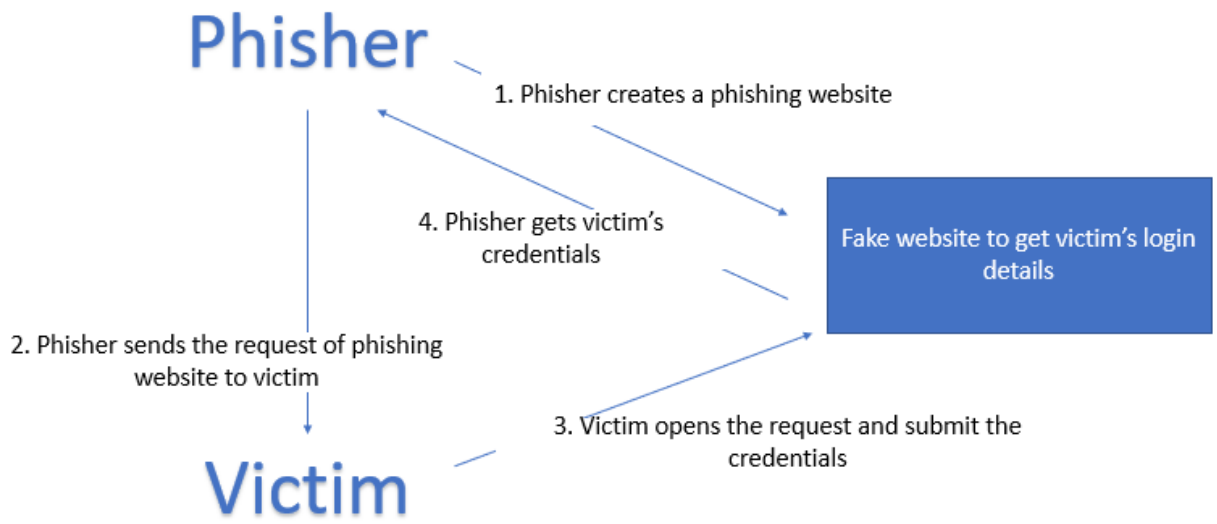


Figure 1.1: The phishing life cycle.

Figure 1.1 shows the life cycle or anatomy of a typical phishing attack which consists of the following steps.

Step 1: The phisher designs a fake website which looks like an existing legitimate website. In creating the fake website, the phisher tries to keep visual similarity between the phishing website to the original one to trick users.

Step 2: The phisher sends an email to the targeted user which includes a link to the phishing website.

Step 3: When the victim opens the email and clicks on the phishing link, he is taken to the phishing website where he is asked to enter personal data. For example, if the phisher has created a phishing website of a bank, then the victim would be asked to enter their banking details.

Step 4: As soon as the victim enters their data on the phishing website, the phisher has the needed details. The phisher may utilize this data for financial or other advantages.

1.2 Project Objectives

The ISOT Lab at the University of Victoria has developed a new phishing detection system that classifies messages (i.e. emails, SMS messages, and tweets) based on the writing style of the sender, also called stylometry. Using stylometry, a unique profile is constructed for each messenger which is known by the message recipient [1]. New messages are compared against the messenger profile in order to establish whether the identity of the sender is genuine or not. This allows phishing attacks to be identified and prevented. The ISOT phishing detection system includes other capabilities such as URL analysis and file attachment structure and format analysis that help detect broader forms of phishing.

Figure 1.2 gives an outline of the ISOT phishing detection system. This system receives as input email messages, and checks for evidence of phishing using four different methods: user identity verification, file attachment verification, message origin verification, and embedded URL checking. It checks the genuineness of the messenger identity, and checks whether key message characteristics match the messaging behavior of the claimed messenger identity with respect to the recipient.

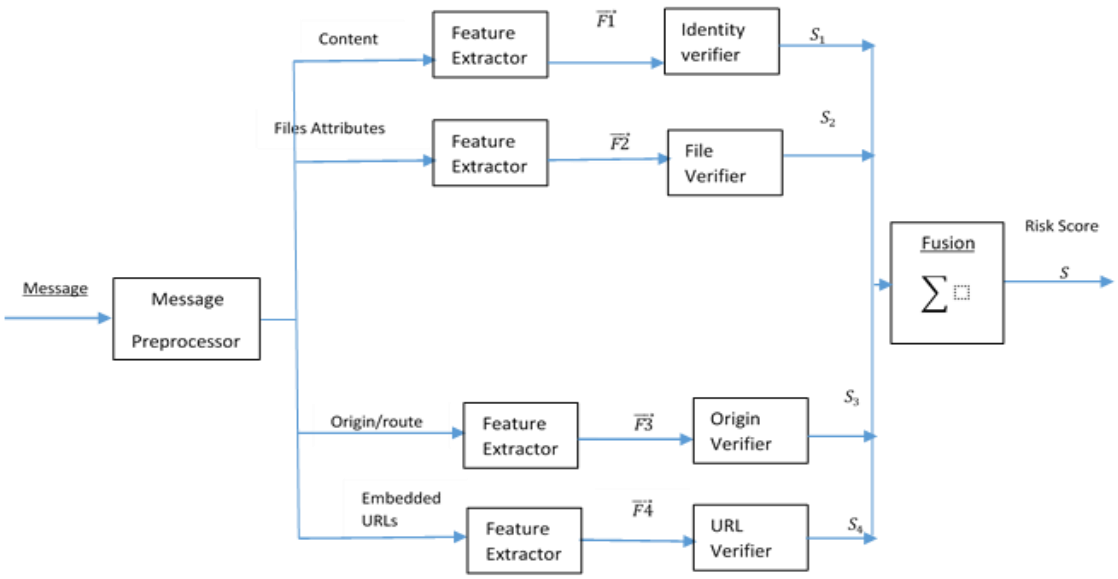


Figure 1.2: The ISOT phishing detection system [1].

The proposed system is used to verify both incoming and outgoing messages. Checking outgoing messages determines whether a legitimate account has been hijacked and is being leveraged to send out phishing messages to victims known by the account holder. Checking incoming messages protects against phishing attacks by establishing whether the sender is genuine and known by the recipient. The current version of the system is being tested using offline datasets including legitimate emails and phishing emails samples. Because the processing of large datasets offline is time consuming and inefficient, it was decided to implement a test utility that would enable online and live testing of the system at a reasonable scale. The goal of this project is to create an environment for testing the abovementioned phishing detection system.

Specifically, the goal of this project is to complete the following tasks.

1. Develop the aforementioned test utility.
2. Create several test email accounts and use the test utility to automatically populate these accounts using existing legitimate, spam, and phishing email datasets.
3. Assess the ability of two popular existing email platforms, namely Gmail.com and Yahoo.com, to filter phishing emails.
4. Conduct functional testing of the software implementation for one of the modules of the phishing detection system, namely, the URL analyzer.

The URL checking module is designed to detect phishing by extracting the features of the URL given from the phishing email using heuristic rules. These heuristic rules are mainly based on the content of the referenced website (in the message) as well as the features of the URL from which it can be decided that a website is legitimate or phishing. These rules are constructed by extracting the features from the URLs and then applying them to if-then-else logic. They help in choosing from three possible outcomes: phishing, legitimate, or suspicious. Test cases were designed for each rule to test whether the code is working correctly or not. In the test cases, the decision was made to detect phishing, legitimate or suspicious using “Assertion method” in JUnit.

1.3 Approach and Tasks

The main idea is to simulate the real world where emails are received from many people, and some may be malicious. By creating such an environment, the test utility can systematically test the accuracy of the phishing detector in real-time. This will determine how good the classification (phishing or legitimate) is.

The first task is to find datasets for legitimate, phishing and spam emails. In this case, we make a distinction between legitimate, spam and phishing. Once the dataset was gathered, the next step is to convert each email sample from the dataset into one with an eml extension. Then several test email accounts were created using Gmail or Yahoo services separately for phishing, spam and legitimate. eml is a file extension for an email message in the format of Microsoft Outlook Express as well as some other email programs.

A python script was designed which reads the messages converted to eml files from the specific path (where eml files are located in the computer) and then sends these files via email (Gmail or Yahoo service) to the specified recipients. The idea behind this is that the phishing emails should be sent to an email account specially created to collect phishing emails, and the same idea is used for legitimate and spam emails. Emails were also directed to specific individuals because one of the goals is to test phishing attacks targeted at individuals. A targeted phishing attack, also called a spear phishing attack, is one the most effective and damaging forms of phishing.

In addition to providing a platform for testing the accuracy of the detectors, a goal of the project was to design test cases for testing the functionality of the software implementation of the URL analyzer, which is one of the modules of the ISOT phishing detection system. For this purpose, the requirements specification of the corresponding module was studied and test cases designed using JUnit.

1.4 Report Outline

The rest of the report is structured as follows.

Chapter 2 provides some background information and highlights the phishing mechanisms and challenges.

Chapter 3 presents the collection of the datasets, the creation of testing accounts and the mechanism for populating these accounts automatically from the datasets.

Chapter 4 presents the testing framework and the testing of the URL analysis module of the ISOT phishing detection system.

Chapter 5 concludes the report and discusses some directions for future work.

Chapter 2: Background on Phishing

Phishing is a cybercrime which employs social engineering and technical fraud to steal sensitive information of online users such as personal information (e.g. social security numbers) and financial account credentials [2]. Phishing is now considered a major threat along with ransomware attacks and data loss. It has become a great security issue in society for private citizens and organizations.

Several solutions have been proposed to protect users against phishing attacks. To warn online users against likely phishing websites, there are many toolbars available for different browsers. However, most of the existing solutions have had limited success as phishing attacks are still thriving.

2.1 Phishing Attack Vectors

Phishing attacks involve different types of attack vectors which are described as follow [15].

Email Spoofing

Email spoofing occurs when an attacker sends an email using a fake or unrelated email address. Email spoofing is possible because the Simple Mail Transfer Protocol (SMTP), which is the main protocol used in sending emails, does not involve by default any authentication process. This provides fertile ground for more sophisticated attacks such as phishing which leverages address spoofing to hide the origin of the phisher.

Fake Social Network Accounts

An attacker can easily target the end user of social networking sites like Facebook, Twitter, and Instagram by creating fake accounts or profiles and befriending potential victims. By using these fake profiles and pretending to be someone they are not (like a famous athlete), attackers can access confidential data disclosed by victim users when they created their accounts.

Malware

Sophisticated phishing attacks often use some form of malicious software to deliver the end goal of the attack, which could be to enslave the user machine (through a botnet), collect private information (e.g. by using key loggers and spyware), or take hostage the user machine (e.g. using ransomware). Most malware falls under the general category of Trojans. This attack is the most deliberate threat to system security. A trojan is a malicious executable program or dynamic loadable library (DLL). For example, clicking on a link or file redirects the user to a fake website. It is code, contained inside an apparently harmless program that is covertly harmful for the system and users. Attackers also use false banking sites to offer lower credit costs or better interest rates than other banks. Victims may be fooled by these false bank offers and provide personal information to the attackers.

2.2 Phishing Motivation

There are several reasons that motivate phishers as outlined below [3].

Financial Gain

Various researchers have indicated that the main victims of phishing attacks are financial institutions. Hence, financial gain is the leading motive for phishing. The most widely used phishing techniques consist of requesting login credentials for victim bank accounts or their credit information.

Identity Hiding

Hiding or anonymization is another leading goal of phishing. The attackers steal victim identities, and either sell the stolen identities to other criminals or use them to commit crimes anonymously.

Fame and Notoriety

Peer recognition is another motive of phishers who want to gain fame and acknowledgment among their fellow cybercriminals.

Malware Distribution

Phishing is sometimes used to spread malware. In this case, attackers distribute the malware by means of phishing messages that are sent in bulk with the goal of enslaving victim machines or using them for ransomware infection.

Harvesting Passwords

Phishing may also be conducted with the goal of harvesting at large scale passwords that can be used by the phisher or sold on the dark web to other cybercriminals. Phishers conduct password harvesting using diverse methods. For instance, by installing key loggers and malware code/script on the victim machines or as browser extensions. Data gathered from victims is either used for financial gain, identity hiding, identity fraud, or sold to criminals for financial gain.

2.3 Phishing Detection Techniques

Various approaches to prevent different phishing attacks are currently available. Some of these approaches are discussed below.

In [4], six different classification algorithms were applied separately to classify emails as legitimate or phishing, namely, C5.0, Naive Bayes, SVM, Linear Regression and K-nearest neighbour. A comparison of these algorithms indicated that the C5.0 algorithm achieves the best accuracy and precision in detecting phishing emails. Next, they developed a technique called R-Boost which involves two phases. First, the emails are classified using the C5.0 algorithm. Second, the emails classified as legitimate in the previous step are reclassified using the ensemble classification method, which selects the best performing algorithm from the algorithms listed above. The main disadvantage in this study is that the false positive rate was not examined which is a very significant metric in this field.

In [5], an email filtering approach was proposed which differentiates between three message types: ham, spam and phishing. They used thirty features in their model, nine of which were

new. They implemented feature extraction and classification as a plugin for the Apache James Server. This approach achieved 97% classification accuracy.

In [6], phishing detection was proposed based on a Bayes classification algorithm. The feature model consists of hybrid features, which are a combination of content-based and behaviour-based features. These features were extracted from both the email header and body. Features extracted from the header include subject-based features, sender-based features, and behaviour-based features. Features extracted from the body include URL-based, keyword-based, form-based and script-based information features. This approach yielded a 96% detection accuracy, with only 4% false positive (legitimate emails incorrectly detected as phishing) or false negative (phishing emails incorrectly detected as legitimate) which is quite high and so is a disadvantage.

Chapter 3: Proposed Test Framework

In this chapter, the test framework which is the main outcome of the project is presented. First, the data collection process for phishing, spam and legitimate emails is explained. Second, the creation of the email accounts for the three distinct categories phishing, spam, and legitimate using Gmail and Yahoo services, is outlined. Last, the design and implementation of the service to read eml files from the collected dataset and send them to the testing accounts is explained.

3.1 Data Collection

The main testing task of the ISOT phishing detection system consists of using an existing dataset to evaluate the efficiency and effectiveness of the proposed system in detecting phishing. Currently, the testing is done offline. The goal this project is to create a utility that will enable online testing of the system.

The test utility involves creating different email accounts and populating them automatically from existing datasets by developing a set of scripts. A separate team will later use these accounts to test the effectiveness of the ISOT phishing detection system.

Three different datasets were collected for the tests: a dataset of phishing emails, a dataset of legitimate emails and a dataset of spam emails. For the phishing dataset, we used 885 phishing emails from the Phishcorpus dataset [7], as shown in Table 3.1. The Phishcorpus dataset is a public email corpus which includes phishing emails collected over several years. They are in mbox format. Mbox stands for MailBox which is the most common format of string email files. The messages in each mailbox are stored as text files starting with the “From” header of the message [8]. Figure 3.1 shows sample emails from the phishing dataset displayed using the mboxview application, which is a utility that supports viewing mbox files.

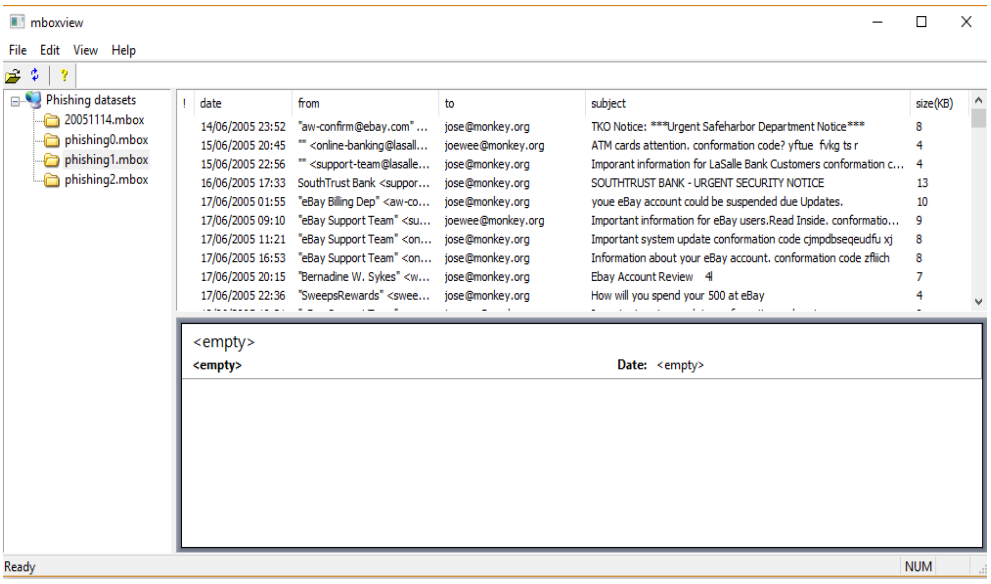


Figure 3.1: The Mbox viewer.

The SpamAssassin dataset [9] was used for the spam dataset. The SpamAssassin dataset is a publicly available email dataset which includes a selection of email messages and is suitable for use in testing spam filtering systems. A subset of the dataset consisting of 528 spam messages was selected as indicated in Table 3.1.

Table 3.1: Email datasets

Dataset	Number of emails	Category
Phishcorpus	885	Phishing
SpamAssassin dataset	528	Spam
Enron Email dataset	446	Legitimate

For the legitimate email dataset, a subset of the Enron email corpus [10] was used. This dataset was originally collected and prepared by the Cognitive Assistant that Learns and Organizes (CALO) project. It contains data from about 150 users, and was made public (posted online), by the Federal Energy Regulatory Commission [10]. The selected legitimate message subset consists of 446 emails as indicated in Table 3.1. The datasets were structured by having each email as a single eml file, and the messages were stored in three different directories depending on their categories, as follows:

i. <directory>/phishing/1.eml, 2.eml, …, n.eml

ii. <directory>/legitimate/1.eml, 2.eml, …, n.eml

iii. <directory>/spam/1.eml, 2.eml, …, n.eml

To convert the files into eml files, the mbx2eml tool is used. It is a free tool used to split emails grouped in mbox file format and store each email in a separate file. The process of converting files grouped in mbox file format to separate eml files is illustrated in Figure 3.2. Figure 3.3 shows the output of the conversion from mbox to eml files.

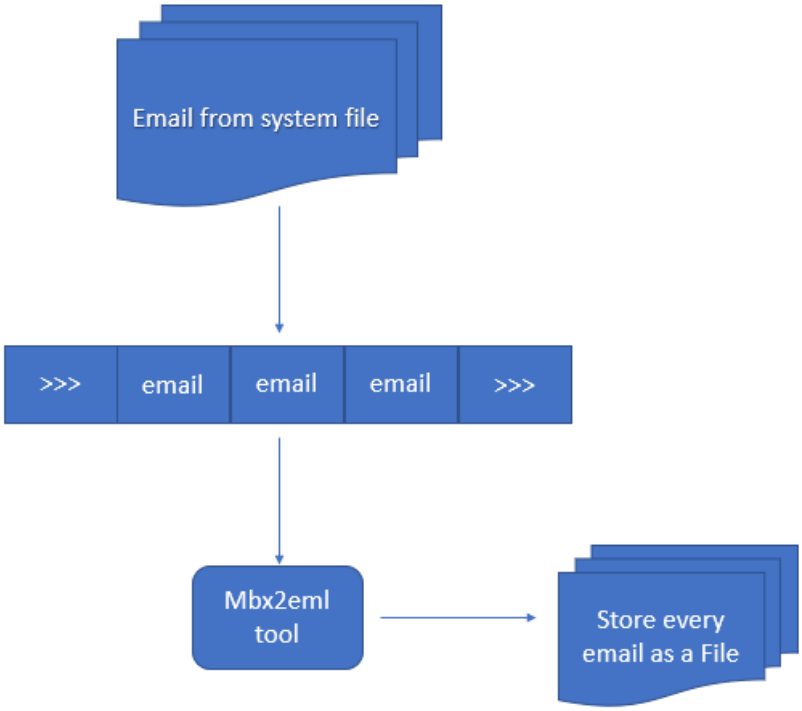
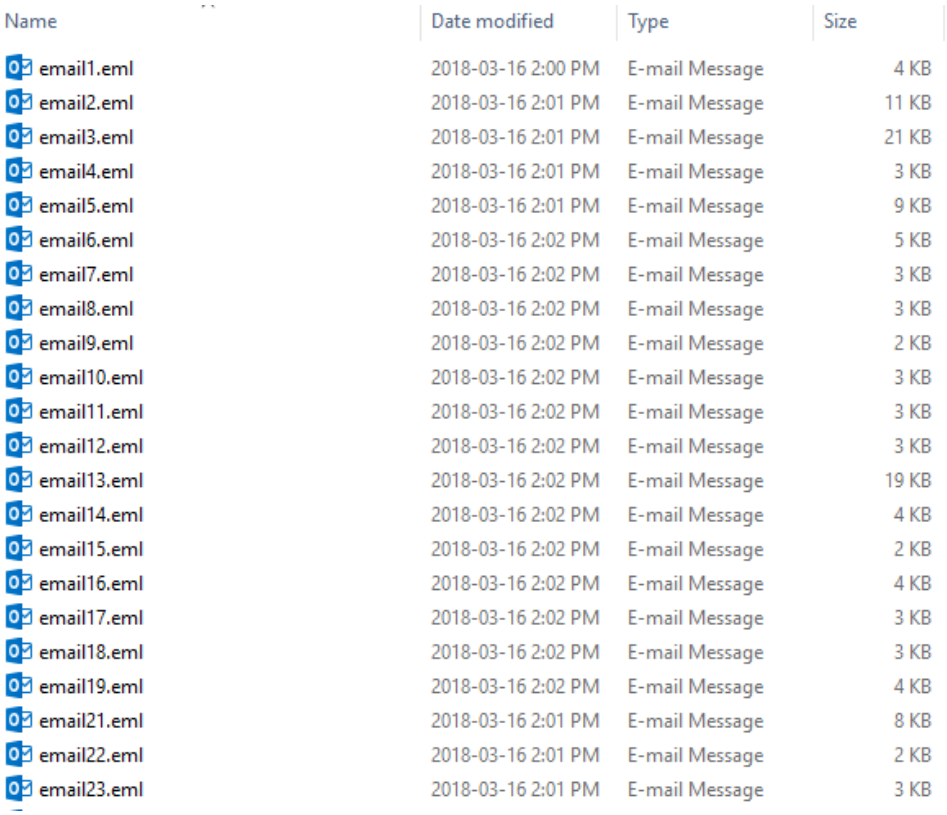


Figure 3.2: The conversion process from mbox folder to eml files.



Name	Date modified	Type	Size
email1.eml	2018-03-16 2:00 PM	E-mail Message	4 KB
email2.eml	2018-03-16 2:01 PM	E-mail Message	11 KB
email3.eml	2018-03-16 2:01 PM	E-mail Message	21 KB
email4.eml	2018-03-16 2:01 PM	E-mail Message	3 KB
email5.eml	2018-03-16 2:01 PM	E-mail Message	9 KB
email6.eml	2018-03-16 2:02 PM	E-mail Message	5 KB
email7.eml	2018-03-16 2:02 PM	E-mail Message	3 KB
email8.eml	2018-03-16 2:02 PM	E-mail Message	3 KB
email9.eml	2018-03-16 2:02 PM	E-mail Message	2 KB
email10.eml	2018-03-16 2:02 PM	E-mail Message	3 KB
email11.eml	2018-03-16 2:02 PM	E-mail Message	3 KB
email12.eml	2018-03-16 2:02 PM	E-mail Message	3 KB
email13.eml	2018-03-16 2:02 PM	E-mail Message	19 KB
email14.eml	2018-03-16 2:02 PM	E-mail Message	4 KB
email15.eml	2018-03-16 2:02 PM	E-mail Message	2 KB
email16.eml	2018-03-16 2:02 PM	E-mail Message	4 KB
email17.eml	2018-03-16 2:02 PM	E-mail Message	3 KB
email18.eml	2018-03-16 2:02 PM	E-mail Message	3 KB
email19.eml	2018-03-16 2:02 PM	E-mail Message	4 KB
email21.eml	2018-03-16 2:01 PM	E-mail Message	8 KB
email22.eml	2018-03-16 2:01 PM	E-mail Message	2 KB
email23.eml	2018-03-16 2:01 PM	E-mail Message	3 KB

Figure 3.3: Examples of eml files generated after conversion using the mbx2eml tool.

3.2 Create Testing Accounts

Email accounts were created to send eml files from the phishing, spam and legitimate datasets. At least one email account was needed for each category. The email accounts were created using both Gmail and Yahoo services. The following email accounts were created for the phishing and spam categories:

- Emailspam249@gmail.com
- Emailphishing249@gmail.com
- Emailspam249@yahoo.com
- Emailphishing249@yahoo.com

For the legitimate category accounts, accounts were associated with 5 users from the Enron dataset. The five email accounts are as follows:

- Email.allen249@gmail.com
- Email.arnold249@gmail.com
- Email.arora249@gmail.com
- Email.badeer249@gmail.com
- Email.baughman249@gmail.com

The reason email accounts were created with the 249 label in each account is to have a marker to identify emails that have been sent from testing accounts only.

After creating the email accounts for the phishing, spam, and legitimate categories, the next step was to send the eml files (not as an attachment, but as an actual email) from the collected datasets and from one of the corresponding test email accounts to emailveritas249@gmail.com. The phishing detection system will filter incoming messages to emailveritas249@gmail.com and segregate emails into phishing or legitimate. This step did not take much time to complete as it required just creating email accounts for the three categories. However, the next step was time consuming as it required reading all eml files from the gathered datasets and sending them to the testing accounts.

3.3 Reading and Sending eml Files

The eml files were separated and email accounts for each category were created manually. The next step was to send them to the test accounts via email. The service to achieve this was implemented in Python. Anaconda distribution is an open source platform enabling fast and easy implementation of Python applications. It is commonly used in industry for developing and testing Python applications. Anaconda navigator is a graphical user interface included in Anaconda which allows one to launch applications and easily manage conda (package manager and environment manager) packages, environments and channels without the need to use command line commands [11].

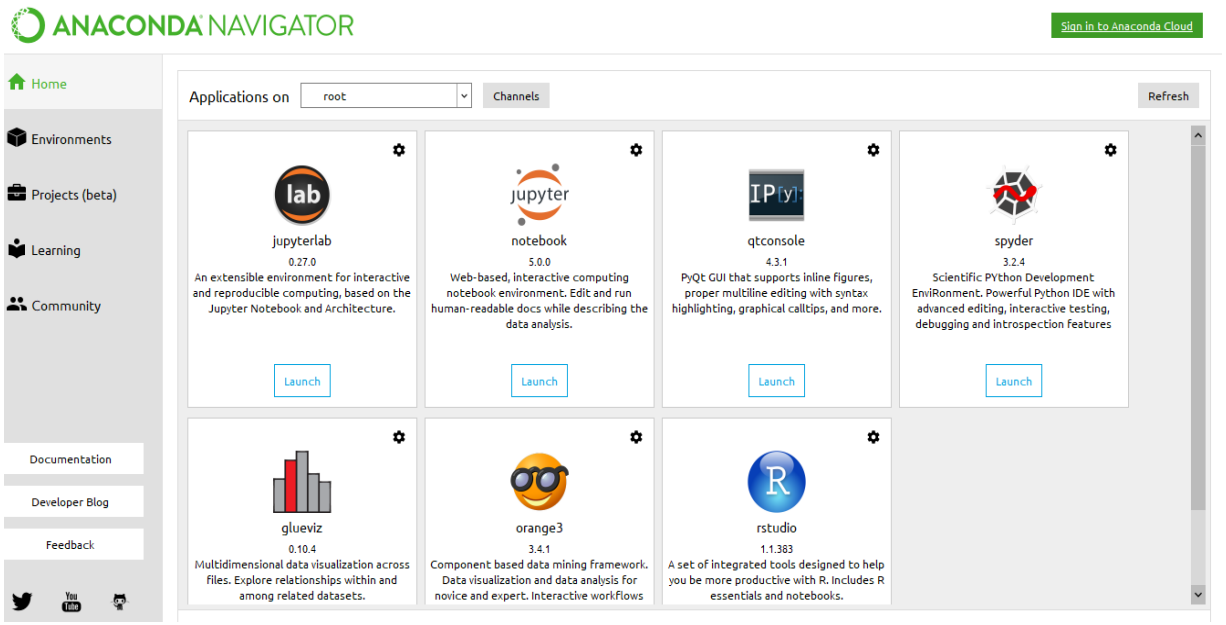


Figure 3.4: The Anaconda Navigator Desktop application in Windows 10.

The Jupyter Notebook web application was launched using Anaconda Navigator application. As shown in Figure 3.4, clicking on the Launch button under the Jupyter Notebook application launches the web application in the web browser as shown in Figure 3.5. It opens as a localhost on the web browser as the program is running on a computer.

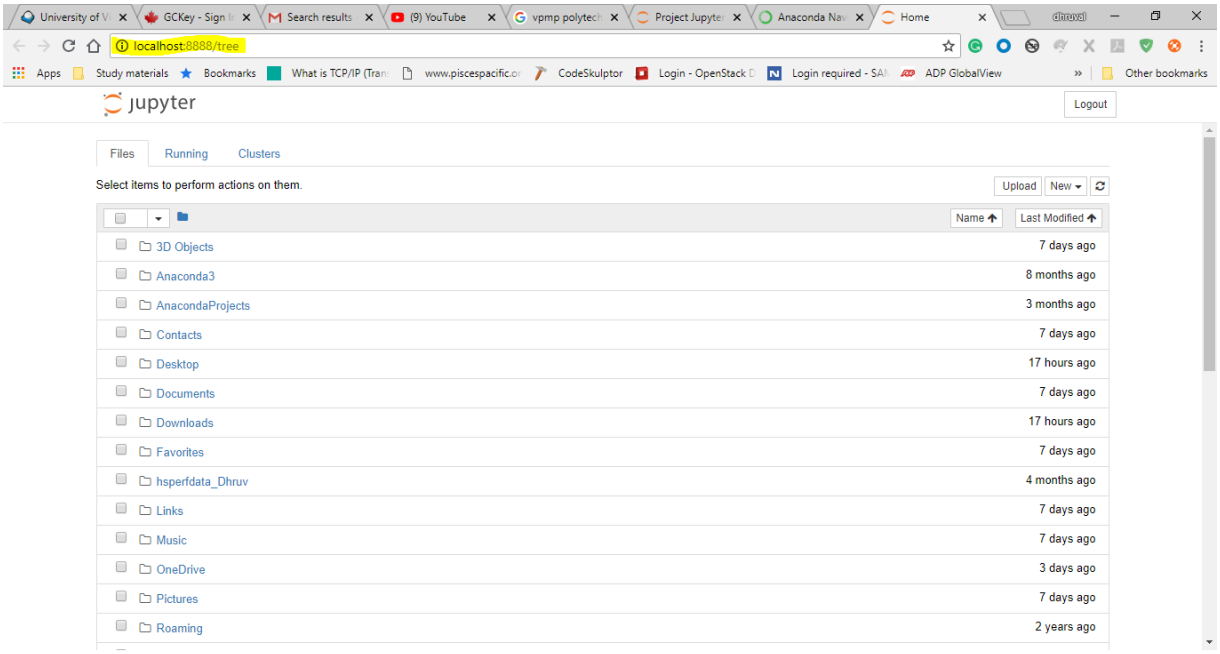


Figure 3.5: Jupyter Notebook launched on a web browser.

To create a new project, click on New in the drop-down menu as shown in Figure 3.6 and select Python from the drop-down menu to create a new python file. This opens in a new tab as shown in Figure 3.7. A flowchart of the eml file reading/sending service implementation is depicted in Figure 3.8.

Figure 3.6: Creating a new project.

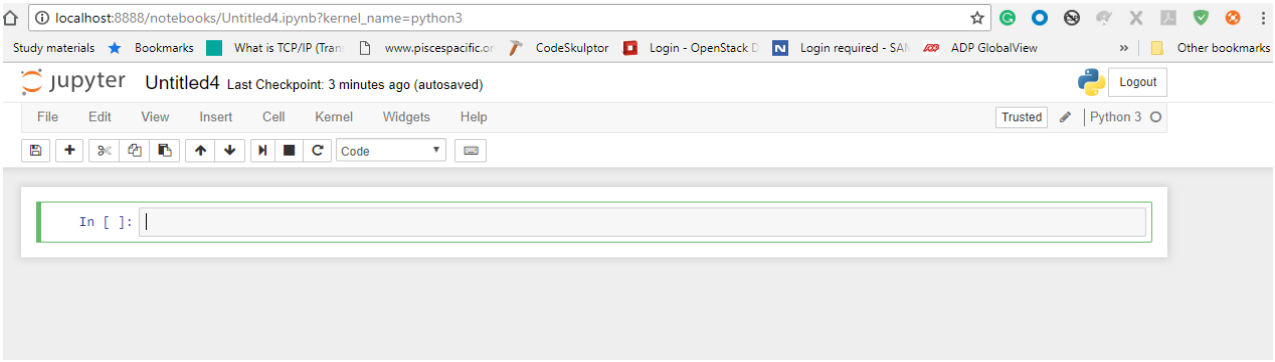


Figure 3.7: New project opens in a new tab.

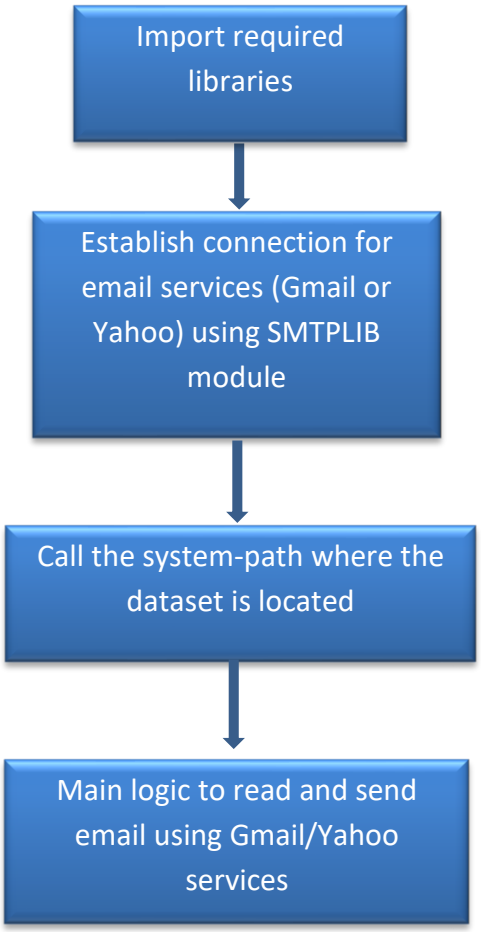


Figure 3.8: Flowchart of the eml file reading/sending service implementation.

This service does not collect the dataset and create email accounts automatically. The email datasets were collected and email accounts created manually, then the service was applied. The connection needs to be established to the email account which is created manually and then the system path where the email dataset is located is called. There is a native library in Python to send emails called smtplib, so that external libraries do not need to be installed. Simple Mail Transfer Protocol (SMTP) is an application-level protocol (Top layer of TCP) which is used to communicate with mail servers from external devices like an email client on a phone (an installed application). SMTP is a delivery protocol so emails cannot be retrieved using this protocol.

As already mentioned, Python comes with the smtplib module which handles parts of the protocol such as establishing connections, authentication, validation and sending emails. Using

this library, there are several ways of creating a connection to the mail server. The unencrypted version can easily be created by just passing the server address, port number, and calling using .ehlo(), which identifies to the SMTP server. Using the above server parameters, emails can be sent over an insecure connection. The server address and port number for Gmail and Yahoo differ for different email services as shown in Table 3.2.

Table 3.2: SMTP server parameters

Email service	Server	Port number
Gmail	smtp.gmail.com	465 or 587
Yahoo	smtp.mail.yahoo.com	465 or 587

When an SMTP connection is secured via SSL/TLS, there are different ways to establish secure SMTP connections in the smtplib module. The first is to create an insecure connection and then upgrade it to TLS using .starttls(). Another way is to initially create an SSL connection using .smtp_ssl().

There are several authentication steps that must be taken before sending emails through Gmail/Yahoo using SMTP. If using Gmail services, Google must allow a connection via SMTP, which is considered a less secure method. This requires turning on “Less secure App access” in the settings as depicted in Figure 3.9. Further, “Allow Apps that use less secure sign in” must be turned on from the Account security settings in Yahoo.

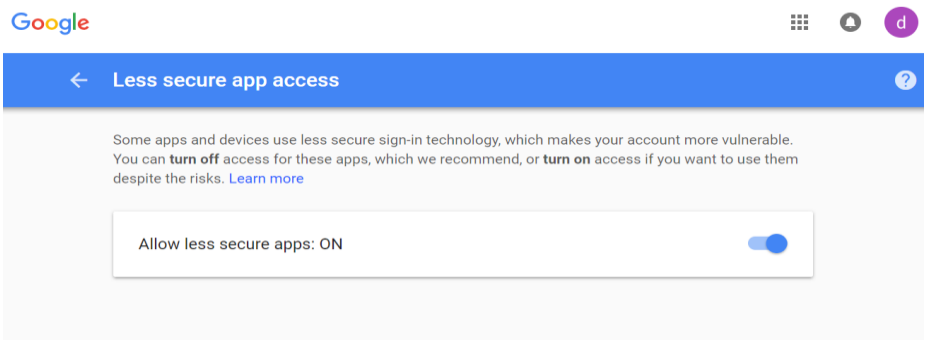


Figure 3.9: Allow access to less secure apps on Google.

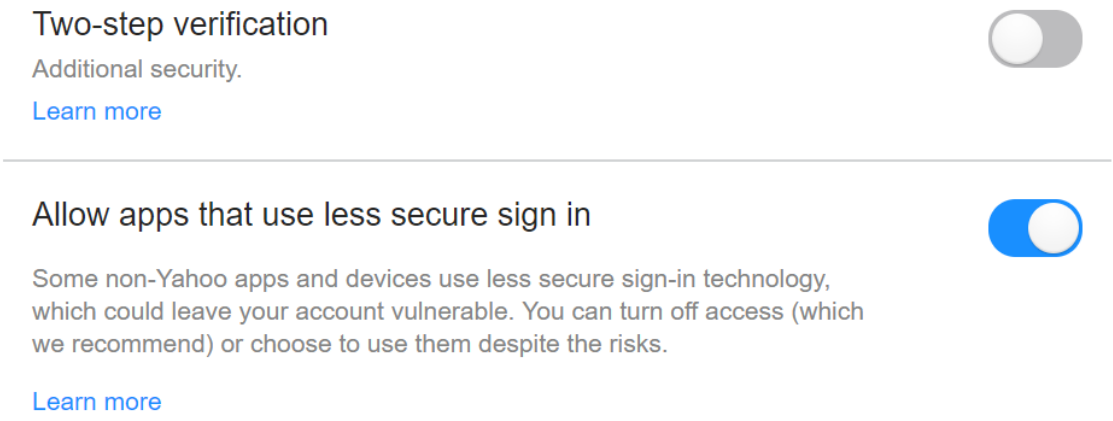


Figure 3.10: Allow access to less secure apps on Yahoo.

After the SMTP connection is set up and authorized by the app with Google/Yahoo, emails can be sent with Gmail/Yahoo using .sendmail(). With this method, the message parameters (sender and recipient email addresses) should be mentioned.

There is a limit to the number of emails that can be sent in Gmail. An error message is generated if the email sending limit is exceeded [12] as follows.

- You have reached a limit for sending email.
- You reached a Gmail sending limit.

After reaching a sending limit, a user must wait up to 24 hours to send an email. The email sending limit per day is 2000 using Gmail (Google services) [12]. If a non-Gmail SMTP service is used such as Microsoft Exchange, Outlook or another client, the SMTP relay service can be configured to route outgoing mail through Google. The daily SMTP relay limit is 550 [13].

3.4 Email Segregation Results for Gmail and Yahoo Test Accounts

As mentioned earlier, a goal of this project is to create the test utility by setting up and populating test email accounts. The evaluation of the phishing detection system using the test utility is beyond the scope of this project. However, the practicality of the test utility was checked by assessing and comparing the ability of the underlying email platforms (i.e. Yahoo

and Gmail) to segregate spam emails. Tables 3.3 and 3.4 provide the filtering results for Yahoo and Gmail, respectively. This shows that both email platforms perform well for spam detection, but many phishing messages escaped detection. Further, Yahoo achieved a better phishing detection rate compared to Gmail as shown in Table 3.5.

Table 3.3: Default filtering results (number of messages) for Yahoo

Message type	Correct classification	False classification	Total number of messages
Legitimate	446	0	446
Spam	528	0	528
Phishing	792	93	885

Table 3.4: Default filtering results (number of messages) for Gmail

Message type	Correct classification	False classification	Total number of messages
Legitimate	446	0	446
Spam	528	0	528
Phishing	699	186	885

Table 3.5: Phishing detection performance for Gmail and Yahoo

	Detection Rate (DR)	False Positive Rate (FPR)
Yahoo	89.5%	0%
Gmail	79%	0%

Since many phishing messages escaped the generic spam filtering, there is a need to run a dedicated phishing detection scheme. The ISOT phishing detector is supposed to run after the default spam filtering by the underlying email platform. The test utility presented in this chapter allows live testing of the ISOT phishing detection system, but this task is out of the scope of the project.

Chapter 4: Partial Test Automation for the Phishing Detector

This chapter presents the unit test cases developed to test one of the modules of the ISOT phishing detection system. The test cases were developed using a test automation framework. An overview of the unit testing and the test automation framework used in the project are provided. Then, several examples of test cases are presented.

4.1 Module Under Testing

One of the modules of the ISOT phishing detection system, the URL verifier, is shown in Figure 1.1. The URL verifier involves subsystems that use different techniques to detect phishing URLs. One of the subsystems uses machine learning classification. Another subsystem uses heuristic rules to detect these URLs. One of the tasks in this project is to generate and execute unit test cases for the heuristic rule-based subsystem. This subsystem implements a collection of rules that extract different features from URLs embedded in a message body and determine whether they are legitimate or phishing.

4.2 Unit Testing

Unit testing is a method by which individual units of source code are tested separately. The main goal of unit testing is to isolate each part of the program and test that the individual parts, methods, or classes are working correctly. It allows automation of the testing process and reduces difficulties in discovering bugs in the system or application. This means that a set of inputs given to a function should return the correct values which can be determined by comparing the returned and expected values.

Advantages of Unit Testing

- Unit testing is done by a developer to test code before integration. Then, issues can be found at an early stage and resolved without impacting other code.
- Unit testing helps in maintaining and changing code because this is done at an early stage of development.
- Unit testing reduces the cost of bug fixes as the bugs are found early in the development cycle. The cost of finding and fixing bugs through unit testing compared to other testing

stages like system testing or acceptance testing is low because it these other forms of testing occur at later stages in software development. Bugs detected at an early stage are easier to fix because bugs detected later can be the result of many changes, and bug traceability can be time consuming and costly.

4.3 Testing Framework

A testing framework is defined as a set of assumptions, concepts, and practices that constitute a work platform or support for testing or automated testing. The testing framework is responsible for defining the format in which to express expectations, execution results and reporting results. There are various testing frameworks and tools for the Java programming language such as JUnit, Selenium, REST-Assured, Robot-framework, Mockito, and TestNG. In this project, the JUnit framework is used.

JUnit is a unit testing framework for the Java programming language. It plays an important role in test-driven development and it is part of the family of unit testing frameworks known as xUnit that originated with JUnit. JUnit is used to set up the test data for a piece of code before it is implemented. This increases the productivity of the programmer and the stability of the code, and reduces the time spent on debugging. The features of JUnit are as follows.

- It is an open source test framework which can be used for writing and running tests.
- It provides annotations to identify test methods.
- It provides assertions for testing expected results.
- It is less complex and time-consuming than the Mockito test framework.

JUnit classes are used in writing and testing in JUnit. Some of the important classes are as follows.

- Assert – contains a set of assertion methods.
- TestCase – contains a test case that defines the fixture to run multiple tests.
- TestResult – contains methods to collect the results of executed test cases.

Almost all the Integrated (Software) Development Environments (IDEs) like Eclipse, NetBeans, and Spring Tool Suite provide JUnit integrations, which allows the writing and running of unit

tests for Java code. The Spring Tool Suite (STS) is used in this project to design the unit tests using JUnit.

4.4 Creating JUnit Test Suite in the Spring Tool Suite

Spring Boot provides several annotations and utilities to help with testing a system or application. Spring boot contains the libraries for JUnit, Mockito, and AssertJ. The main window of STS after importing the project into STS is shown in Figure 4.1. To create a test suite class, click menu File > New > Other… > Java > JUnit, as shown in Figure 4.2. Then, select JUnit Test Suite as highlighted in Figure 4.2 and click “Next >”. This opens a dialogue box as shown in Figure 4.3. Insert the name of the test and hit "Finish". After clicking Finish, the new test class is created which can be used to write test units.

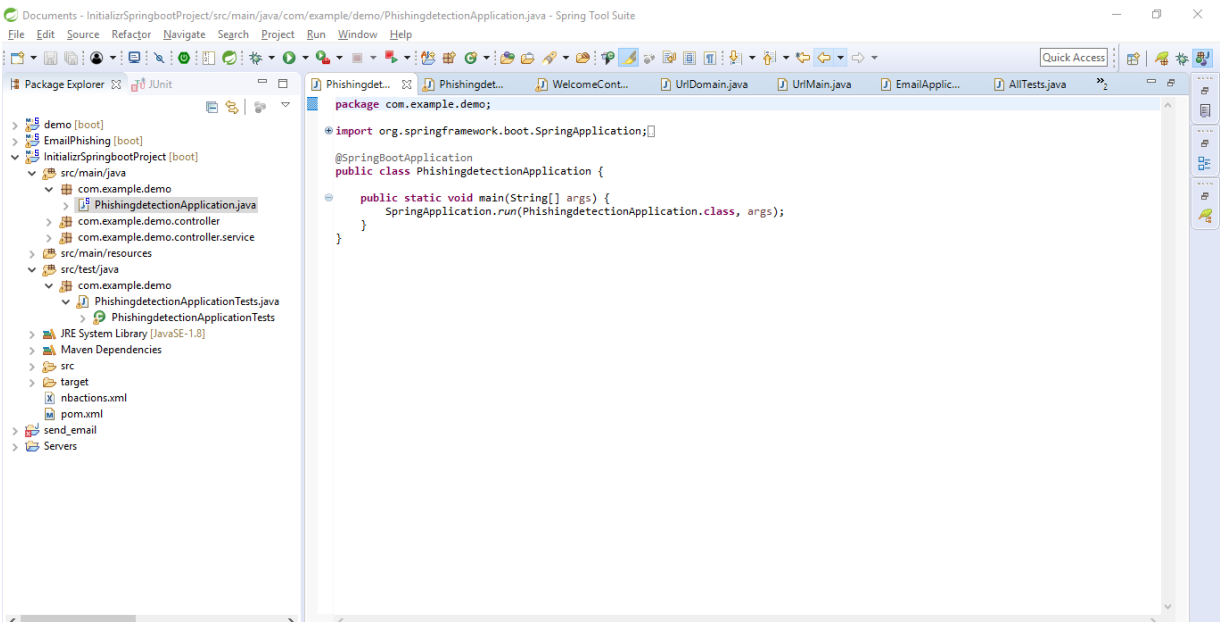


Figure 4.1: STS after importing the project.

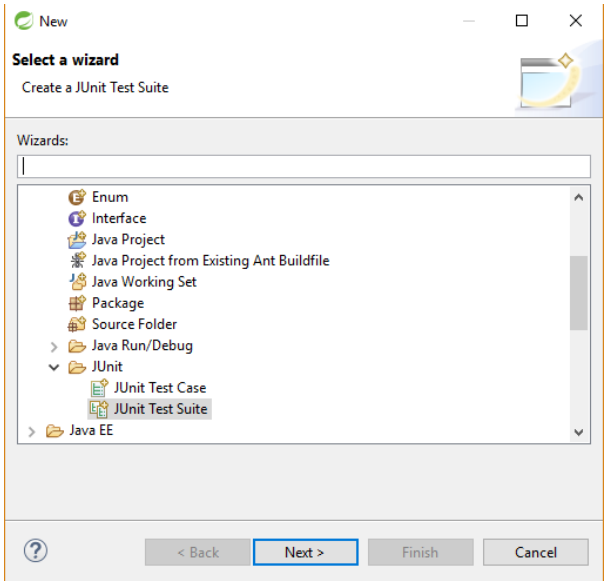


Figure 4.2: Create a new test suite class.

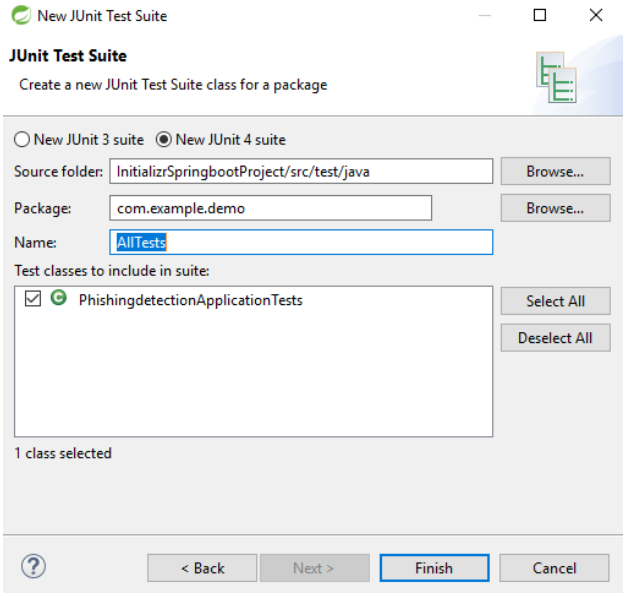


Figure 4.3: Create the name of the test suite.

4.5 Examples of Test Units with Heuristic Rules

In this section, examples of test units written for specific heuristic rules are explained along with the code.

Example 1: Test Unit for the Alexa Rank Page Rule

The Alexa rank page of a website is extracted using the Alexa rank page web API. The rank page gives the number of visitors to the website. If there are many visitors, the website is classified as legitimate and if there are few visitors, the website could be phishing [14]. The rule is expressed as follows. If the Alexa rank of a website is less than or equal to 100,000 then it is legitimate, otherwise it is phishing.

```
public class PhishingdetectionApplicationTests {

    @Test
    public void testAlexarankpage_rank() throws Exception {

        String url = "https://www.gmail.c@o.in.bc/fhhhc/jjvjsjc/jhdshjvds/jbvsj!bhjsvb/hjvbshbvhdsbvd/jhbvjsdsakl/fdfdfdss";
        //String url = "https://www.google.com/";
        //String url = " ";
        Alexarankpage ar = new Alexarankpage(url);

            double[] ranking = ar.alexRank(url);
            for(int i=0; i<3 ; i++) {
                System.out.println(ranking[i]);
            }
            double[] expected = {1.0,0.0,0.0};
            assertNotNull(ranking);
            assertArrayEquals(expected, ranking, 1);


        }
```

Figure 4.4: Test unit for the Alexa Rank Page rule.

The public class name is the name given when creating the JUnit test suite as shown in Figure 4.4. Figure 4.5 depicts the test unit written for the Alexa Rank Page heuristic rule. JUnit supports annotations for configuring tests and extending the framework. Here, the “@Test” annotation, as shown in Figure 4.4, has been used which denotes that a method is a test method.

JUnit provides many assertion methods. Here, only assertNotNull() and assertArrayEquals() are used. The assertNotNull() method tests a single variable to determine if it is not null. In this example, it checks the value of the variable “ranking” which should not be null. If the value is not null then the test passes, otherwise it fails. The assertArrayEquals() method tests whether two arrays are equal i.e, if the two arrays contain the same number of element and the elements in the arrays are equal. The elements must also be in the same order. First, the expected array is created. Second, the “ar.Alexa rank()” method is called, which is the method

to be tested. Finally, the result of the “ar.Alexa rank()” method call is compared to the expected array. If the arrays are equal, assertArrayEquals() proceeds without any errors. If the arrays are not equal then an exception is thrown, and the test aborted. Any test code after the assertArrayEquals() will not be executed. The third parameter in assertArrayEquals() is the value of delta. Delta is the floating-point tolerance for comparisons.

The Alexarankpage(URL) class returns {Phishing, Suspicious, Legitimate} = {1.0, 0, 0} if the output of the URL is phishing. In this example, a random URL which is phishing was used, so the expected array is {1.0, 0, 0} and the result of the test class should also be the same. Thus, the assertArrayEquals() test is passed if the value of both arrays are the same. The result for this case is shown in Figure 4.5.

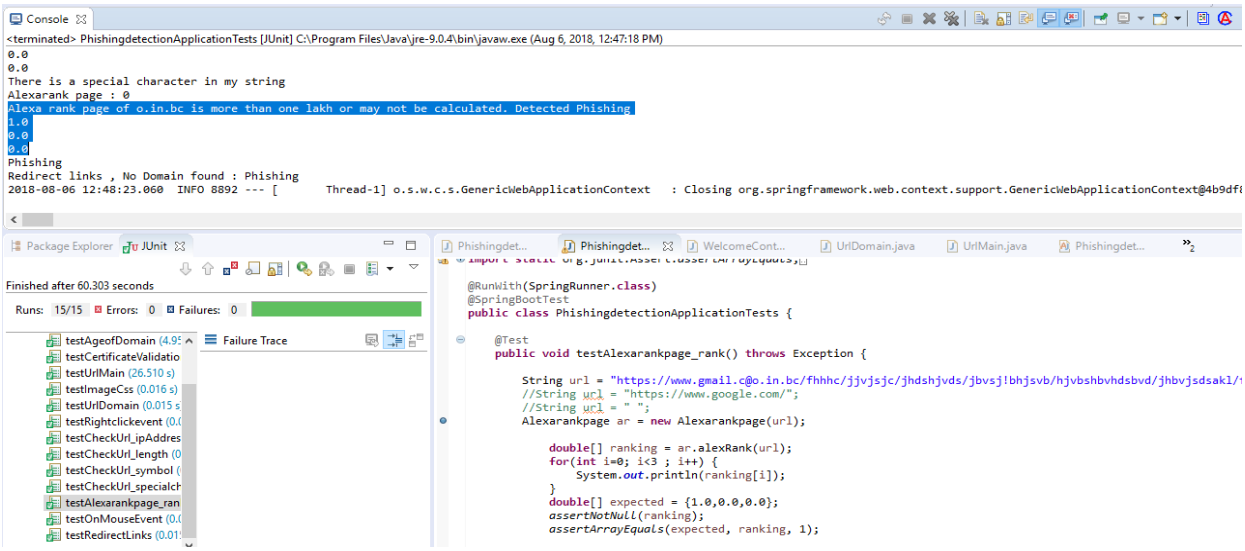


Figure 4.5: The result of the test unit detected as phishing.

Figure 4.6 shows the results of the test unit detected as legitimate and as failed. The assertNotNull() method value of the ranking is not null as shown in Figure 4.6. The Alexarankpage(URL) class returns {0.0, 0.0, 1.0} as the URL is detected as legitimate. However, the expected value is {1.0, 0.0, 0.0} so the values of the arrays are different. This is why assertArrayEquals() fails as shown in Figure 4.6.

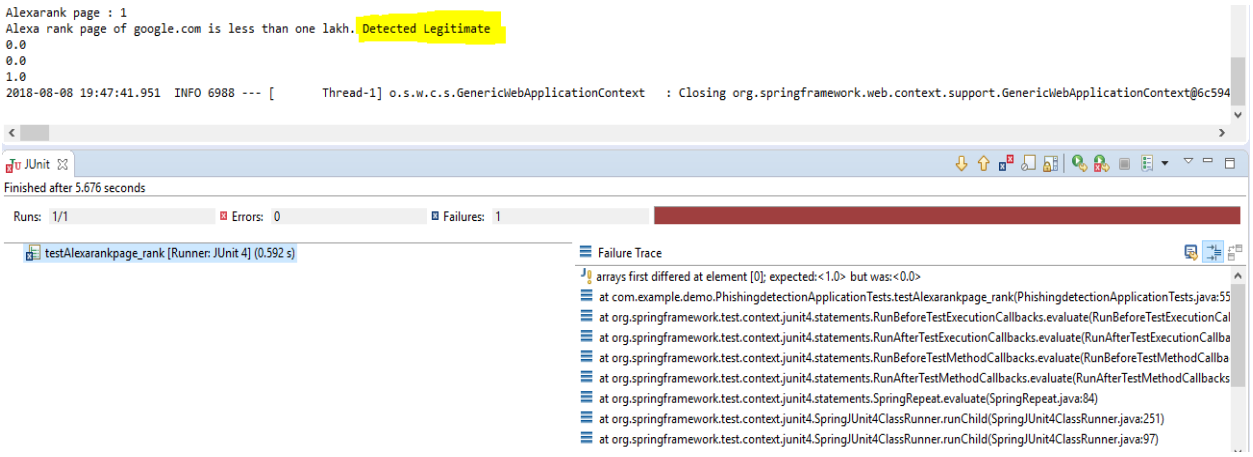


Figure 4.6: The result of the test unit detected as legitimate.

Example 2: Test Unit for Age of Domain Rule

The domain name is extracted from the URL and the domain creation date is checked by performing a query on the WHOIS database [14]. This rule is expressed as follows. If the Age of Domain is more than or equal to 6 months, it could be legitimate. Otherwise, there is some likelihood that it is phishing. The test unit for Age of Domain heuristic rule is shown in Figure 4.7. In this example, the same annotation "@Test" and assertions to test this unit were used as in the previous example.

```java
    @Test
    public void testAgeofDomain() throws Exception {
        String url = "https://www.google.com/";
        Ageofdomain age = new Ageofdomain(url);

        double[] domain = age.domaianAge(url);
        for(int i=0; i<3; i++) {
            System.out.println(domain[i]);
        }

        double[] expected = {0.0, 0.0, 1.0};

        assertNotNull(domain);
        assertArrayEquals(expected, domain, 1);

    }

```

Figure 4.7 Test Unit for the Age of Domain rule.

The result of assertNotNull() should not be null. In this case, it is not null as shown in Figure 4.8 which means this test is passed. The expected value and the result of “age.domainAge(URL)" are the same, which means the values of the arrays are the same, so assertArrayEquals() is passed as shown in Figure 4.8.

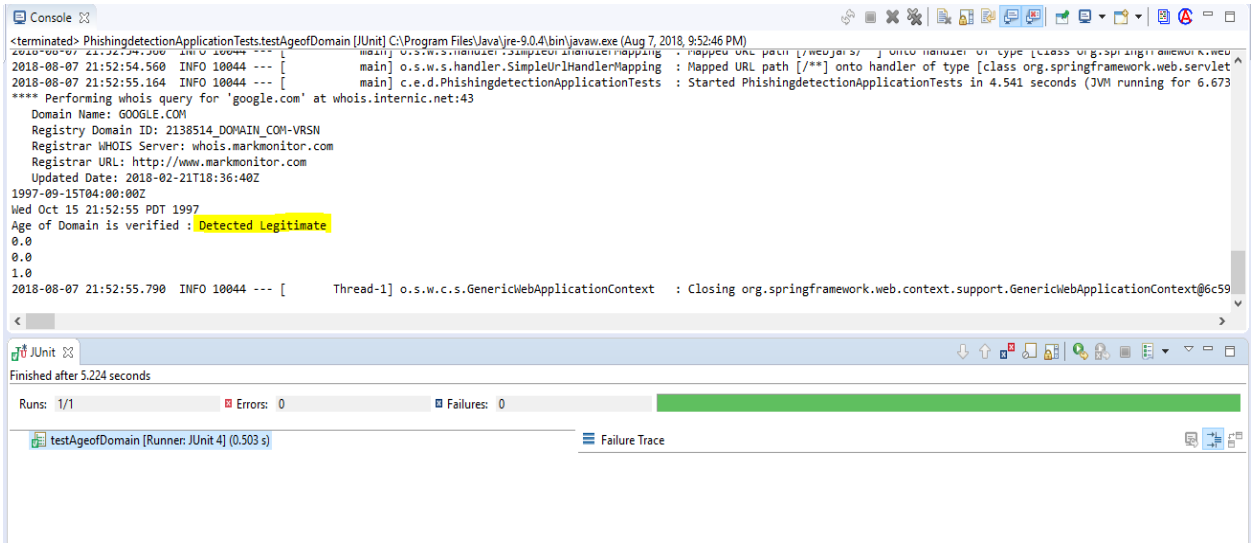


Figure 4.8: The result of the test unit detected as legitimate.

Example 3: HTTPS with the Certificate Validation Rule

A URL has either the HTTP or HTTPS protocol as a header. If the header is HTTPS, then the SSL certificate should be verified. This involves checking various attributes of the certificate like the date to check whether it is expired or not, whether it is self-signed, and whether its public key is verified or not [14]. This rule is expressed as follows. If the certificate is active and not self-signed, then it is classified as legitimate. Else if it is self-signed or the public key verification fails, then it is classified as suspicious. Else if the certificate is expired and self-signed or public key verification fails, then it is classified as phishing.

```
60  @Test
61  public void testCertificateValidation_date() throws CertificateNotYetValidException, MalformedURLException, CertificateException, NoSuchAlgorithmException
62
63      //String url = "https://www.gmail.c@o.in.bc/fhhhc/jjvjsjc/jhdshjvds/jbvsj!bhjsvb/hjvbshbvhdsbvd/jhbvshdbvhsdbvhdsb/jbhsvdhv/";
64      String url = "https://www.google.com/";
65      CertificateValidation cert = new CertificateValidation(url);
66
67      double[] date = cert.getCertificateValues(url);
68
69      for(int i=0; i<3; i++) {
70          System.out.println(date[i]);
71      }
72
73      double[] expected = {0.0, 0.0, 1.0};
74
75      assertNotNull(date);
76      assertArrayEquals(expected, date, 0.0);
77  }
```

Figure 4.9: Test for the Certificate Validation rule.

Figure 4.9 shows the test unit for the Certificate Validation heuristic rule. This test unit also uses the same annotation and assertions methods as previous test units. In this case, the legitimate URL is input, and assertNotNull() is passed as it does not have a null output as shown in Figure 4.10. The input URL is detected as legitimate as its certificate is active, so the expected value is set to legitimate. The result of .getCertificateValues(url) and the value of the expected array are the same, so assertArrayEquals() is passed as shown in Figure 4.10.

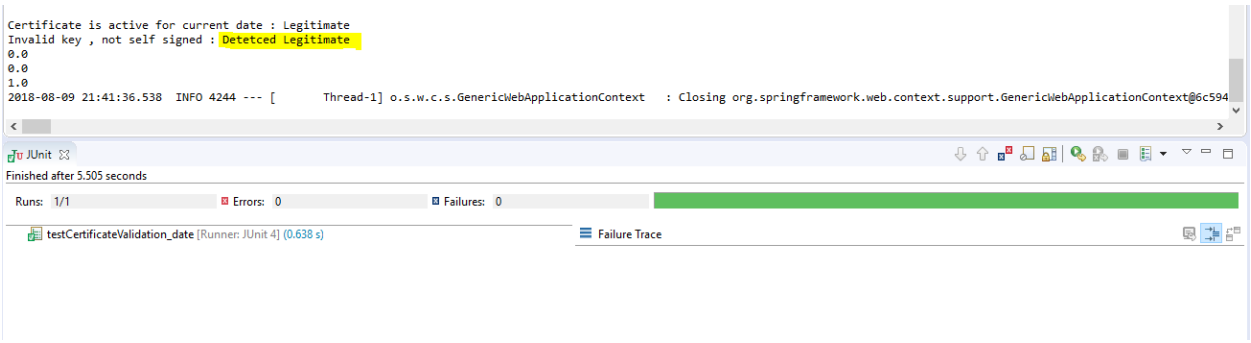


Figure 4.10: The result of the test unit detected as legitimate and the tests are passed.

4.6 Test Cases and Test Results

For the URL verifier module in the ISOT phishing detection system, phishing detection is done using 16 different heuristic rules. In the system, 11 main classes were defined, and 1 class was defined with 5 sub-classes. This covers all 16 heuristic rules. To test the system, 15 test cases were designed using assertion methods. Ten test cases were designed to test the 10 main classes and 5 test cases were designed to test the class with five sub-classes. The getter-setter method was used to test the class with five sub-classes. The getter method is used to obtain or retrieve a variable value from the class, and the setter method is used to store the variables.

The class with five sub-classes checks the 5 different heuristic rules, length of the URL, number of dots and slashes in the URL, presence of @ symbols in the URL, IP address mentioned in the URL, and the presence of special character such as ',', '_', ';' in the URL. Initially, only a single test case was created for the class with five sub-classes, but it was failing as this class has five methods as shown in Figure 4.11. After applying the getter-setter method, all the test cases passed without any issues. The test results are shown in Figure 4.12. assertNotNull() is used to check if the input URL is not empty, and assertArrayEquals() is used to compare the result from the detection method with the expected result.

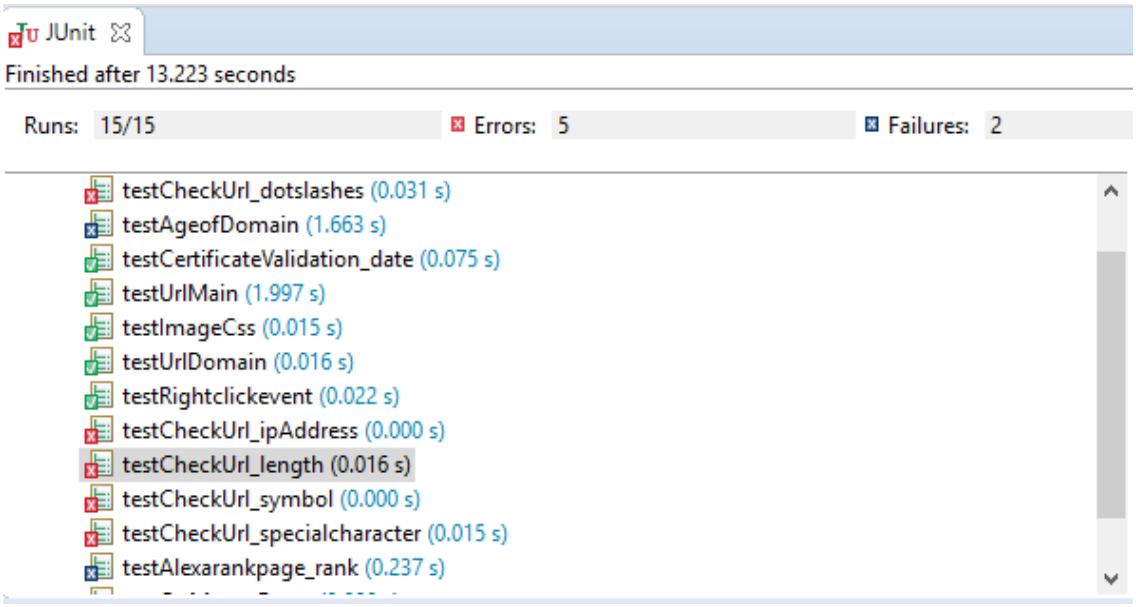


Figure 4.11: Some test cases are failed.

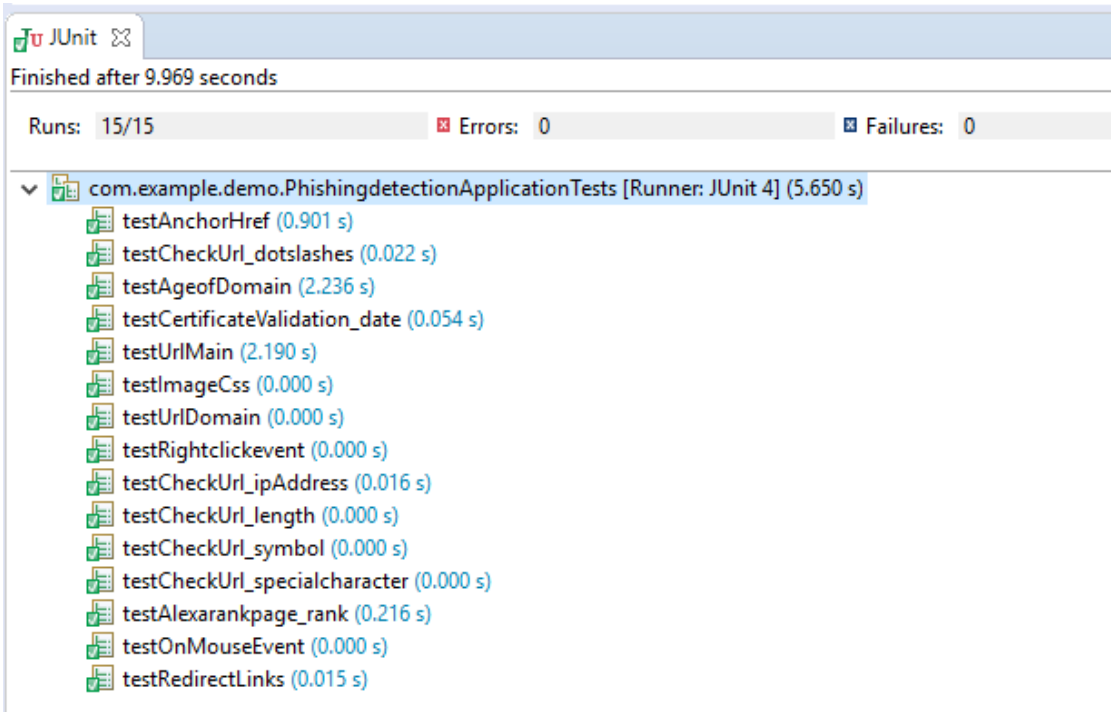


Figure 4.12: All test cases are passed.

Chapter 5: Conclusion and Future Work

Phishing websites are being designed to trick people into submitting credentials to access private information and assets by making the sites look like legitimate websites. Phishing attacks through URLs embedded in emails or SMS messages are one of the major issues faced by Internet users because of the huge number of online transactions performed daily. Phishing attacks cause losses to organizations, customers and Internet users.

In this project, testing utilities were developed to support test automation for different aspects of the ISOT phishing detection system. In particular, two kind of test utilities were developed.

1. A mechanism to populate live test email accounts from different datasets, including spam, phishing, and legitimate emails, allowing online testing of the ISOT phishing detection system.
2. Automated test cases were used to unit test one of the modules of the ISOT phishing detection system.

In this report, the data collection process for phishing, spam and legitimate emails was discussed. Further, the ISOT message security system was explained, and the design and implementation of the service to read the eml files from the collected datasets and send them to the test accounts was explained. The testing was done using the JUnit framework to check the functionality of the different heuristic rules of the system. The results of the unit tests were verified using assertion methods in the JUnit framework.

Future Work

In the future, optimization can be done in the test units and these units can be made fully automated using Robot-Framework. This is important if more heuristics rules are included in the detection system. If the URL length is very long i.e. more than a million characters, then the system may crash. To prevent this situation, a timeout feature can be added when determining the URL length.

References

[1] I. Traore, M. L. Brocardo, “Secure Personalized Trust-based Messages Classification System and Method”, US Provisional Patent Application No. 62/569,250, Filed: 2017.

[2] J. Allen, L. Goman, M. Green, P. Ricciardi, C. Sanabria, S. Kim, “Social Network Security Issues: Social Engineering and Phishing Attack”, Center for Strategic and International Studies, Pace University, 2012.

[3] S. Sharma, S. Karla, “A Comparative Analysis of Phishing Detection and Prevention Techniques”, International Journal of Grid and Distributed Computing, 2016.

[4] F. Toolan, J. Carthy, “Phishing Detection Using Classifier Ensembles”, in IEEE eCrime Researchers Summit, 2009.

[5] W. N. Gansterer, D. Polz, “Email Classification for Phishing Defense”, in Advances in Information Retrieval. Springer, 2009.

[6] I. R. A. Hamid, J. Abawajy, “Hybrid Feature Selection for Phishing Email Detection”, in Algorithms and Architectures for Parallel Processing. Springer, 2011.

[7] https://monkey.org/~jose/phishing/

[8] N. Moradpoor, B. Clavie, B. Buchanan, “Employing Machine Learning Techniques for Detection and Classification of Phishing Emails”, in Science and Information Computing Conference, 2017.

[9] http://spamassassin.apache.org/old/

[10] https://www.cs.cmu.edu/~./enron/

[11] https://anaconda.org/anaconda/anaconda-navigator

[12] https://support.google.com/a/answer/166852?hl=en

[13] https://support.google.com/a/answer/2956491#sendinglimitsforrelay

[14] S. Jeeva, E. Rajsingh, "Intelligent Phishing URL Detection Using Association Rule Mining", Human-centric Computing and Information Sciences, 2016.

[15] S. Gupta, A. Singhal, A. Kapoor, “A Literature Survey on Social Engineering Attacks: Phishing Attacks”, in International Conference on Computing, Communication and Automation, 2016.