



ÇUKUROVA UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

Phishing E-Mail Prevention System: Confirmed Mail

Author:

Bedirhan Budak

Supervisor:

Lect. PhD Elif Emel Fırat

Türkiye, 2023

Contents

List of Figures	III
List of Codes	V
1 Introduction	2
2 How Does Confirmed Mail Work?	6
2.1 Cmail Page	7
2.1.1 Start Button	7
2.1.2 Stop Button	8
2.1.3 Format Check	9
2.1.4 Login Check	10
2.2 Logs Page	12
2.3 Settings Page	12
2.3.1 Service Adjustment	13
2.3.2 Inbox Cleaning	14
2.3.3 Preferences	16
2.4 Info Page	18
2.5 Other Features	19
2.5.1 Receiving an E-Mail	19
2.5.2 Tray Menu	21
2.5.3 Minimize & Quit Buttons	21
3 Source Code of Confirmed Mail	23
3.1 cmail.py	23
3.1.1 Variable Definitions	23
3.1.2 start() Function	24
3.1.3 del_info() Function	29
3.1.4 del_email() Function	30

3.2	authenticator.py	31
3.3	design.py	32
3.3.1	Form & Tab Widget	32
3.3.2	Minimize & Quit Buttons	32
3.3.3	Cmail Tab	33
3.3.4	Logs Tab	34
3.3.5	Settings Tab	34
3.3.6	Info Tab	36
3.4	main.py	37
3.4.1	Cmail Class	37
3.4.2	Connections	37
3.4.3	Functions	40
3.4.4	LoopThread Class	50
3.4.5	Show the Application Window	51
4	Experimentation	52
4.1	Experiment Settings	52
4.2	Questions Asked	52
4.3	Participant Demographics	53
4.4	Qualitative Analysis	55
5	Limitations and Future Work	56
5.1	Runtime and Logic Errors	56
5.2	Interface Enhancements	56
5.3	Source Code Enhancements	57
5.4	Integration Into E-Mail Services	58
6	Conclusion	59
	References	60

List of Figures

2.1	Marking in Gmail	6
2.2	Cmail Page	7
2.3	Started Message Box	7
2.4	Started Log Record	8
2.5	Running Status	8
2.6	Stopped Message Box	8
2.7	Stopped Log Record	9
2.8	Stopped Status	9
2.9	Invalid Format Status	9
2.10	Invalid Format Message Box	10
2.11	Invalid Format Log Record	10
2.12	Login Failed Status	11
2.13	Login Failed Message Box	11
2.14	Login Failed Log Record	11
2.15	Logs Page	12
2.16	Settings Page	13
2.17	Service Adjustment	13
2.18	Inbox Cleaning	14
2.19	DEL Information Message Box	14
2.20	Deleted Message Box	15
2.21	DEL Information Log Record	15
2.22	Cancelled Message Box	15
2.23	DEL Not Confirmed Message Box	16
2.24	DEL Not Confirmed Log Record	16
2.25	Preference Settings	17
2.26	Notifications Off Log Record	17
2.27	Notifications On Log Record	17

2.28	Auto DEL On Log Record	18
2.29	Auto DEL Off Log Record	18
2.30	Info Page	19
2.31	New E-Mail Log Record	19
2.32	New Information E-Mail Log Record	20
2.33	New Confirmed Mail Message Box	20
2.34	New Confirmed Mail Log Record	20
2.35	Tray Menu Options	21
2.36	Minimize & Quit Buttons	21
2.37	Minimized Message Box	22
2.38	Minimized Log Record	22
2.39	Quit Message Box	22
3.1	From and Tab Widget in Qt Designer	32
3.2	Cmail Tab in Qt Designer	33
3.3	Logs Tab in Qt Designer	34
3.4	Settings Tab in Qt Designer	35
3.5	Info Tab in Qt Designer	36
4.1	How Many E-Mail Accounts Do You Actively Use?	53
4.2	How Frequently Do You Check Your E-Mails?	53
4.3	Do You Check the Legitimacy of Incoming E-Mails?	54
5.1	Comparison of Application Size by Resolution	57
5.2	Example Confirmation Mark	58

List of Codes

3.1	Definitions of the Global Variables	23
3.2	Connection and Searching	24
3.3	Parsing New E-mails	25
3.4	Confirming an E-Mail	25
3.5	Creating Confirmation Code	26
3.6	Extracting the Username of Sender	26
3.7	Extracting the Content of Incoming E-Mail	27
3.8	Creating Response Message and Sending E-Mail	28
3.9	Deleting Information E-Mails	29
3.10	Deleting Not Confirmed E-Mails	30
3.11	Generating Confirmation Codes	31
3.12	Cmail Class Definition	37
3.13	Signal Connections	37
3.14	Button Connections on Cmail Page	38
3.15	Button Connections on Logs Page	38
3.16	Button Connections on Setting Page	38
3.17	Button Connections on Info Page	38
3.18	Button Connections in Minimize & Quit Buttons	39
3.19	Tray Menu	39
3.20	clicked_on_start()	40
3.21	clicked_on_stop()	41
3.22	check_format()	42
3.23	login_error()	43
3.24	logs_email()	43
3.25	logs_info()	44
3.26	logs_cmail()	44

3.27 del_info()	45
3.28 del_email()	45
3.29 check_notification()	46
3.30 check_deleteInfo()	46
3.31 Information Buttons	47
3.32 Minimize to Background	47
3.33 Exit	48
3.34 Creating Log Record	48
3.35 Information Message Box	48
3.36 Question Message Box	49
3.37 Constructor of LoopThread	50
3.38 run()	50
3.39 stop()	51
3.40 delete_info()	51
3.41 Show the Application Window	51

Abstract

Social engineering attacks continue to pose a significant and persistent cyber security threat, particularly through e-mail based phishing attacks. In spite of that, there is still no robust method to prevent e-mail based phishing attacks. Unlike conventional messaging, e-mail addresses and content of e-mails can both be completely forged. Even if users confirm the e-mail's content and sender, there is always a possibility that the e-mail is fake. The only way to be sure is to contact the sender of the e-mail. In order to prevent this risk, we developed the Confirmed Mail (Cmail) project, which automates security measures by ensuring that incoming e-mails are confirmed by the sender rather than relying solely on user judgment. When a new e-mail arrives in the user's inbox, Cmail automatically responds with an e-mail containing a unique code to inform the sender. If the sender replies to this response with the unique code, the e-mail is confirmed. Through this approach, Cmail completely eliminates the risk of forged e-mails, significantly enhancing overall security against e-mail based phishing attacks. In this thesis, we will discuss the usage of the project, its source code, participant experiences, and future works.

Keywords: Confirmation, e-mail, phishing, prevention, social engineering

CHAPTER 1

Introduction

Technology is evolving every single day. However, threats to the security of sensitive data are also increasing. There are a variety of methods to steal sensitive data, such as malware attacks, man-in-the-middle attacks, injection attacks, social engineering, and more. These methods can be used by attackers to steal confidential data from a targeted person.

The majority of attacks to steal sensitive data are becoming easier to prevent thanks to the rapid development of technology. For instance, an antivirus software protects from malware attacks, and even recently, artificial intelligence techniques have been used in malware analysis [1]. Similarly, there are some strategic methods that offer practical prevention mechanisms against injection attacks [2].

There are also many methods that have been developed against social engineering attacks [3, 4, 5]. On the other hand, there are many different types of social engineering attacks, such as smishing, baiting, scareware, pretexting, phishing, and spare phishing [6]. Hence, the prevention methods used for social engineering are not adequate for all these types.

Over the past few years, more and more people have been directly impacted by social engineering attacks. The biggest reason for this is that our activities are increasingly shifting to online platforms. This leads to an increased presence of people online, opportunity for attackers. For this reason, social engineering attacks have become the most common type of attack [7]. Therefore, it is necessary to develop new techniques to prevent social engineering attacks.

Social engineers use psychological manipulation to trick users into making security mistakes or providing sensitive information [8]. One of the most common types of social engineering attacks is phishing [9]. In phishing attacks, fake texts that trick users into disclosing private information may be sent via e-mail. This is due to a security weakness in the protocols used to send e-mails. Because of this security weakness, attackers can use any e-mail address to send e-mails to anyone they want. Thus, e-mail based phishing attacks can be very deceptive.

E-mail is a common method of communication for both business and personal matters, so the protection of its functioning and integrity is a matter of widespread importance. At the same time, e-mail has proven to be a significant threat vector, providing an avenue for various attacks, including malware, phishing, and spam. In addition, the use of e-mail, if not properly directed and managed, can pose other risks, with the potential for compromised privacy and reputational damage [10].

Most phishing attacks start with an e-mail that pretends to come from an trusted source. Phishing e-mails often look professional and are no different from e-mails from reliable sources [11]. A new generation of phishing attacks called spear phishing, is becoming increasingly popular and may make it even more difficult for users to distinguish between legitimate and spoofed e-mails. They often contain information that is easily recognizable to the target group, which may in turn convince recipients that the e-mail is legitimate [12].

One study adopted a between-subjects design to examine the impact of individual characteristics and system characteristics on phishing detection. Three hundred and ninety-eight participants worked to identify whether forty e-mails were legitimate or forged. The results showed that systems with feedback and high reliability improved users' e-mail identification performance. Users with a high tendency to trust automation are at higher risk of phishing [13].

People who frequently send and receive official or personal e-mail are more at risk of phishing attacks than others because there is no assurance that any e-mail they receive is secure. Although there are many different techniques to ensure their security, there is no advanced method to prevent e-mail phishing attacks. This makes e-mail phishing attacks one of the most common types of phishing attacks [14]. Because of this, a robust method must be developed to prevent e-mail phishing.

Some groups have proposed some methods to confirm incoming e-mails, such as authorship verification. Authorship verification can be checked using stylometric techniques through the analysis of the linguistic styles and writing characteristics of the authors. While stylometric techniques can yield high accuracy rates for lengthy documents, identifying an author becomes challenging for shorter texts, particularly when faced with a substantial number of potential authors [15].

Similarly, there are several methods for detecting phishing e-mails with machine learning. One study compared the prediction accuracy of these several different machine learning methods for predicting phishing e-mails [16]. Even more methods exist that use machine learning to detect phishing e-mails at a very high rate [17], but these methods might not always be accurate. The linguistic styles and spelling of e-mail senders can be imitated, allowing attackers to manipulate the confirmation.

Another method is Server-Side Verification. In this method, e-mail clients could sign messages with the sender's private key, and the recipient's e-mail client could easily compare the signature with the public key associated with the e-mail domain [12]. However, this method is also not sufficient because it is also possible that attackers can compromise keys or manipulate communication to trick recipients, as in the case of the study [18]. Since it is impossible to confirm whether incoming e-mails are actually from that person.

Taking this information into account, it is difficult to confirm the consistency of online communication between two sources. The sources can be any person or any system. Phishing e-mails can deceive their targets by making themselves appear to come from this source. To fully confirm the authenticity of any incoming e-mail, the sender of the e-mail must confirm that they sent the e-mail. By doing this, the sender of the e-mail can be identified as the owner of that e-mail address. In fact, this method could be inspired by various systems used in real life.

The three-way handshake method for connecting to the internet is a reliable way of establishing a connection between the server and the client. The connection is established in three steps, and these steps are reciprocal between the server and the client [19]. Basically, the client sends a request to the server, the server responds by confirming that it can accept the request and is available, and then the client sends a packet back. Thus, the connection is established.

Another verification method that is commonly used in most places is two-step verification, which is used to verify the identity of the logged-in user. This method basically sends a verification code to a verification tool belonging to the user who owns the account to verify an access authorization. If the user enters this code, it authorizes access. The two-step verification method offers a convenient solution for efficient and fast data validation. For this reason, it can be used in many different areas, such as the verification of big data [20].

In this thesis, we propose a new method for the detection of phishing e-mails by confirming the authenticity of incoming e-mails. This method calls for the step-by-step confirmation of incoming e-mails by the user and the sender, which must be done in order. It is based on concepts such as two-step verification and the three-way handshake. After sending an e-mail, the sender expects a confirmation e-mail with a unique code, similar to two-step verification. Afterwards, the sender replies to this e-mail with the unique code and the application confirms the reply, similar to a three-way handshake, and then Cmail marks the original e-mail as a confirmed non-phishing e-mail.

By developing Cmail, we aim to provide users with an automated security measure and protect them against e-mail phishing attacks based on carelessness and deception. Cmail provides a higher level of assurance of e-mail authenticity by offering a proactive solution that does not leave e-mail security solely to the user, but involves the active participation of the original sender in the authentication process. In the following sections of this thesis, we will explore the implementation and evaluation of Cmail in detail, highlighting its effectiveness and potential impact on mitigating e-mail based phishing threats.

The rest of the paper is organized as follows: Chapter 2 describes in detail how to use the Cmail application and its user interface. Chapter 3 provides a detailed description of the source code of the Cmail application. Chapter 4 describes the feedback received after testing the project with a group of participants. Chapter 5 discusses the issues that were encountered during project development along with suggestions for future improvements. Finally, Chapter 6 presents the conclusion.

CHAPTER 2

How Does Confirmed Mail Work?

This chapter describes in detail how the Confirmed Mail works, and how to use its user interface.

Confirmed Mail (Cmail) is basically an e-mail security system developed to prevent phishing attacks by confirming that incoming e-mails are actually coming from the sender. It checks whether incoming e-mails are sent from legitimate addresses or not. In order to do this, it seeks the user's e-mail address to check for recent e-mails. If a new e-mail arrives or if there is a new unread e-mail in the inbox, Cmail sends a response e-mail to the sender of the incoming e-mail with a unique confirmation code. Once the sender replies to this response e-mail, it is called "*Information e-mail*". It can be deleted by Cmail after it has been read by selecting it in the settings. Cmail checks it, and if the confirmation code in the reply matches the code in the sent response, the first incoming e-mail is marked, as shown in Figure 2.1. Thus, this e-mail is confirmed, and called "*Confirmed Mail*".

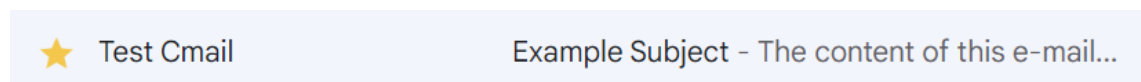


Figure 2.1: Marking in Gmail

The Cmail application has a graphical interface where the background processes can be controlled. This interface consists of four main pages. These are: Cmail, Logs, Settings, and Info.

2.1 Cmail Page

This page, shown in Figure 2.2, is the main page of the application. Cmail is opened with this page when it is executed.

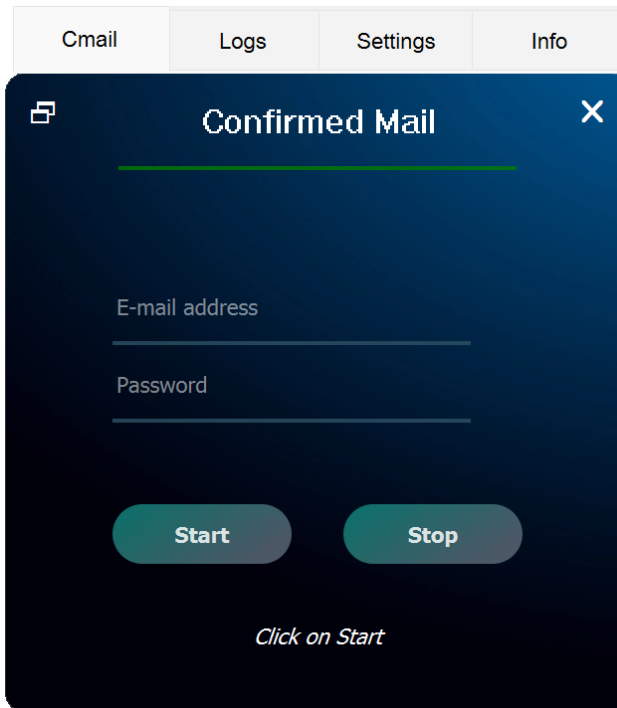


Figure 2.2: Cmail Page

This page has two line edits where the user enters their e-mail address, and password. After filling them in, the user clicks on the "*Start*" button to run Cmail.

2.1.1 Start Button

The "*Start*" button creates a message box with the text "*Cmail has been started*", and the title "*Confirmed Info*", as shown in Figure 2.3:

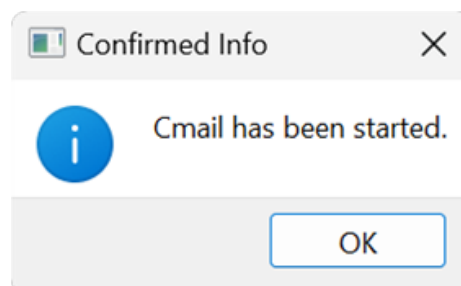


Figure 2.3: Started Message Box

This event will also result in "*Cmail has been started.*" being written in the log, as shown in Figure 2.4:

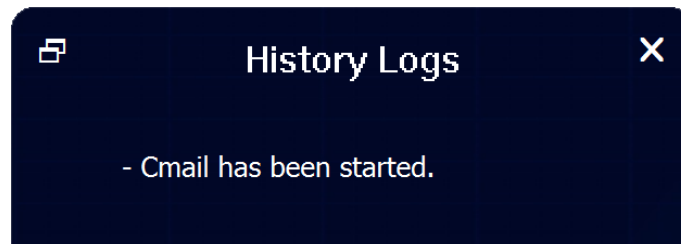


Figure 2.4: Started Log Record

There is a status label at the bottom of the page. It says "*Cmail is running*" when Cmail is running, as shown in Figure 2.5:

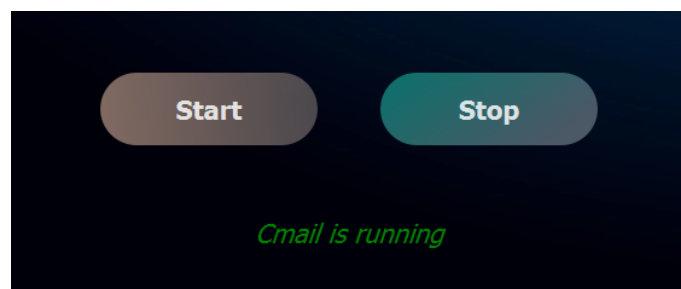


Figure 2.5: Running Status

2.1.2 Stop Button

If user clicks on the "*Stop*" button, Cmail stops working, and creates a message box with the text "*Cmail has been stopped*", and the title "*Confirmed Info*", as shown in Figure 2.6:

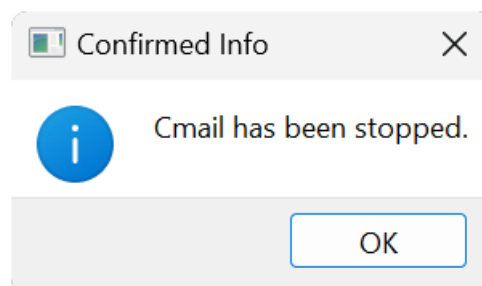


Figure 2.6: Stopped Message Box

This event will also result in "*Cmail has been stopped.*" being written in the log, as shown in Figure 2.7:

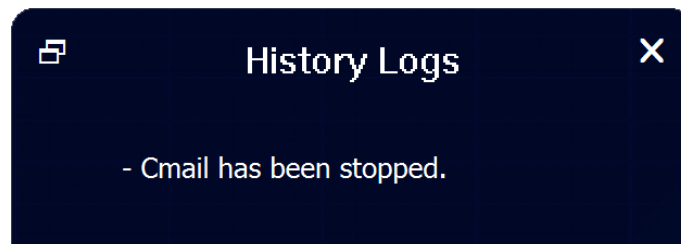


Figure 2.7: Stopped Log Record

The status label says "*Cmail has stopped*" when Cmail is stopped, as shown in Figure 2.8:

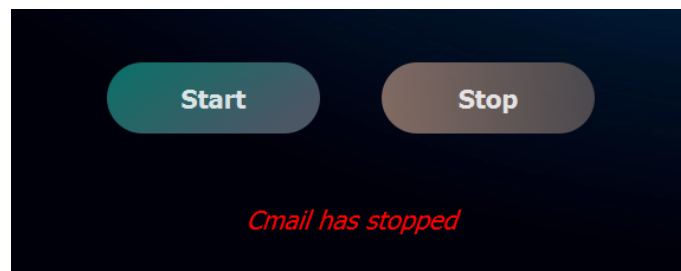


Figure 2.8: Stopped Status

2.1.3 Format Check

If the user enters an e-mail address or password in an invalid format, "*Invalid format*" will be written in the label under the password line, as shown in Figure 2.9:

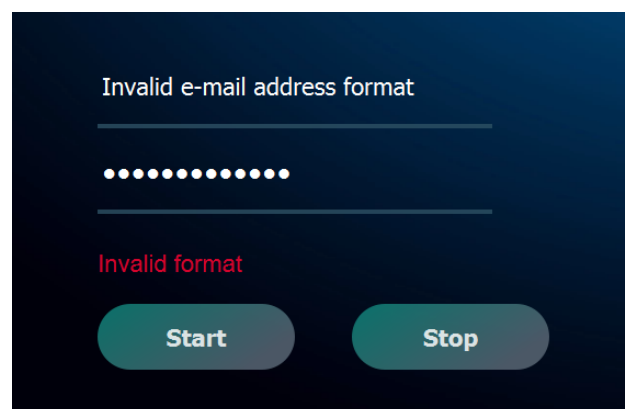


Figure 2.9: Invalid Format Status

It creates a message box with the text "*Invalid e-mail address or password!*", and the title "*Error*" as shown in Figure 2.10:

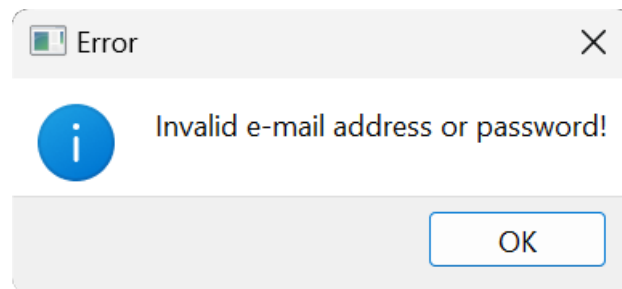


Figure 2.10: Invalid Format Message Box

This event will also result in "*Invalid e-mail address or password entered.*" being written in the log, as shown in Figure 2.11:

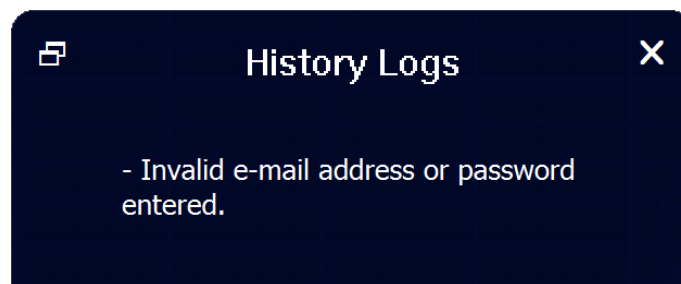


Figure 2.11: Invalid Format Log Record

2.1.4 Login Check

If the e-mail address, and password entered by the user do not match even though they are in the correct format, "*Login failed*" will be written in the status label, as shown in Figure 2.12. Therefore, Cmail is stopped.

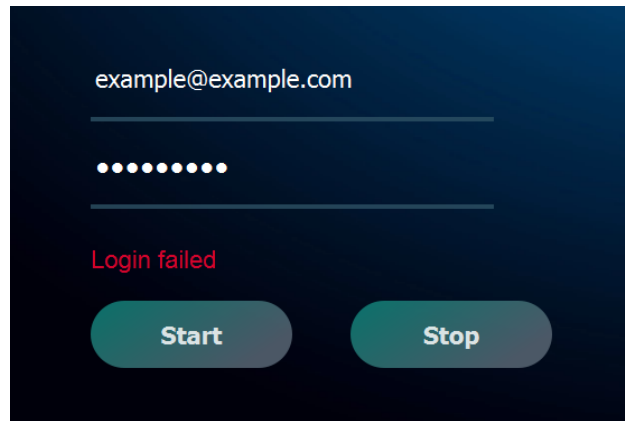


Figure 2.12: Login Failed Status

It creates a message box with the text "*The e-mail or password is incorrect!*", and the title "*Error*", as shown in Figure 2.13:

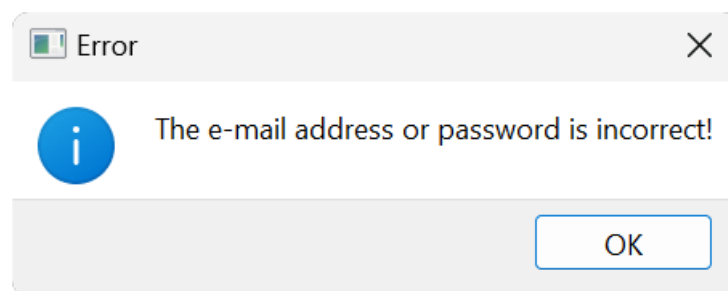


Figure 2.13: Login Failed Message Box

This event will also result in "*Failed to login!*" being written in the log, as shown in Figure 2.14:

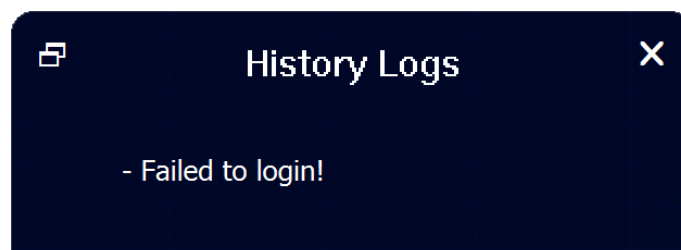


Figure 2.14: Login Failed Log Record

2.2 Logs Page

Log record are displayed in the page shown in Figure 2.15:

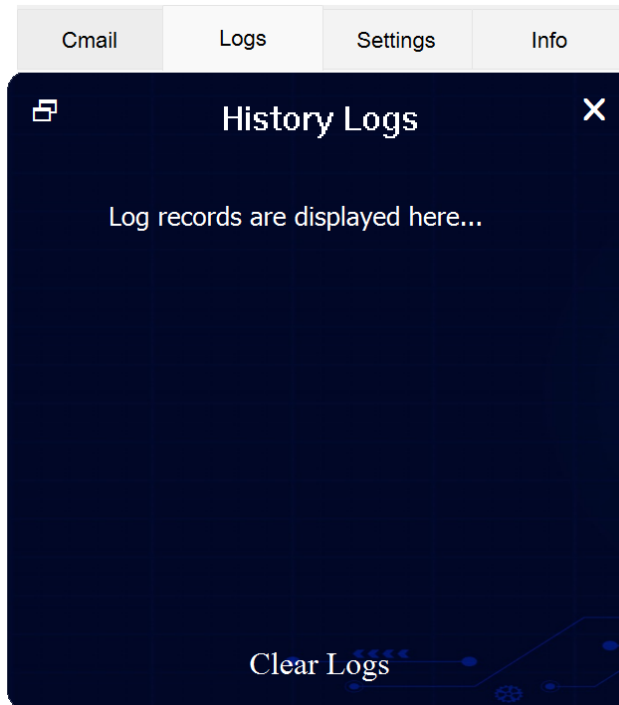


Figure 2.15: Logs Page

Some events create logs. These events are described in their respective fields of application. On this page, all the created logs are displayed. Additionally, there is a "*Clear Logs*" button at the bottom of the page. It clears the displayed log history.

2.3 Settings Page

All settings for Cmail are on the page shown in Figure 2.16:

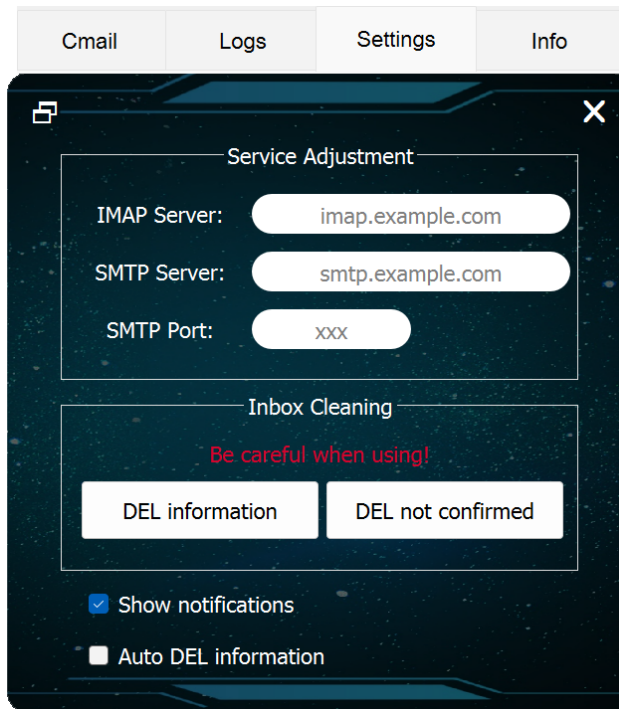


Figure 2.16: Settings Page

Settings are divided into several sections.

2.3.1 Service Adjustment

"*Service Adjustment*" section, shown in Figure 2.17 is used to adjust the services required to read, and respond to incoming e-mail to the e-mail address. The user has to fill in this section with the details of the e-mail address provider they used before starting to run Cmail. Otherwise, the application will not work properly.

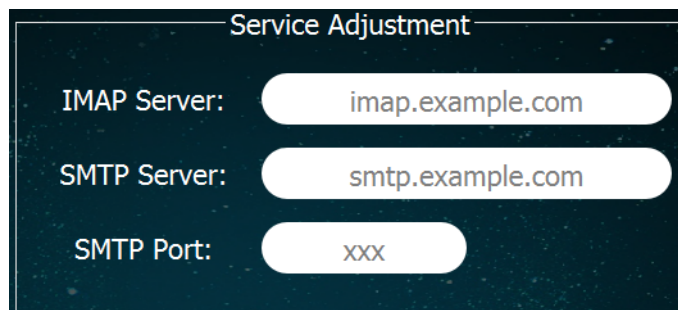


Figure 2.17: Service Adjustment

2.3.2 Inbox Cleaning

"*Inbox Cleaning*" section, shown in Figure 2.18 is optionally available to keep the inbox clean. There are two buttons here, and it is recommended to be careful when using these buttons. Because it can delete all the e-mail in inbox.

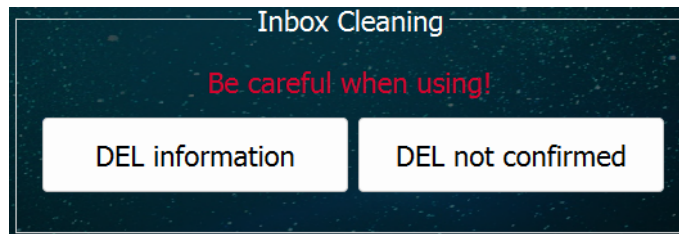


Figure 2.18: Inbox Cleaning

DEL Information

"*DEL Information*" button, shown in Figure 2.18 is used to delete the information (reply) e-mails in the inbox.

When user clicks on this button, it creates a message box with the text "*Are you sure you want to delete information e-mails?*", and the title "*Confirmation Message*", as shown in Figure 2.19:

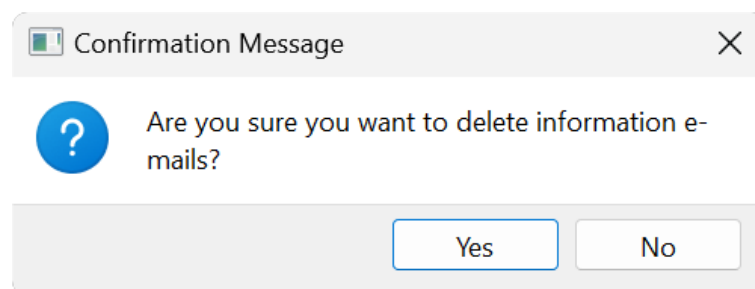


Figure 2.19: DEL Information Message Box

If user clicks "Yes", information (reply) e-mails are deleted from inbox, and it creates another message box with the text "*Deleted successfully*", and the title "*Confirmed Info*", as shown in Figure 2.20:

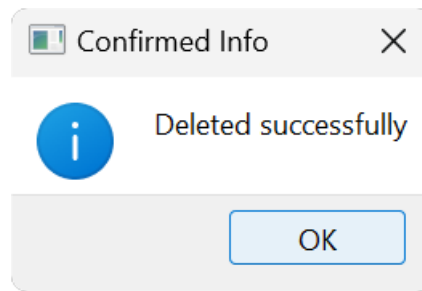


Figure 2.20: Deleted Message Box

This event will also result in "*Information e-mails deleted.*" being written in the log, as shown in Figure 2.21:

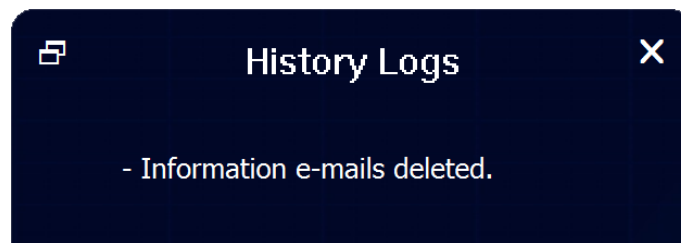


Figure 2.21: DEL Information Log Record

If user clicks "*No*", the process is cancelled, and it creates another message box with the text "*Cancelled by user*", and the title "*Confirmed Info*", as shown in Figure 2.22:

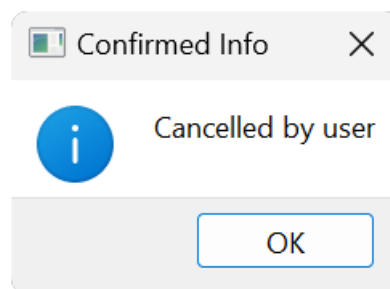


Figure 2.22: Cancelled Message Box

DEL Not Confirmed

"*DEL Not Confirmed*" button, shown in Figure 2.18 is used to delete all e-mails in the inbox except Confirmed Mails.

When this button is pressed, all e-mails except the marked (flagged) e-mails are deleted. If user click on this button, it creates a message box with the text "*Are you sure you want to delete all except Confirmed Mails?*", and the title "*Confirmation Message*", as shown in Figure 2.23:

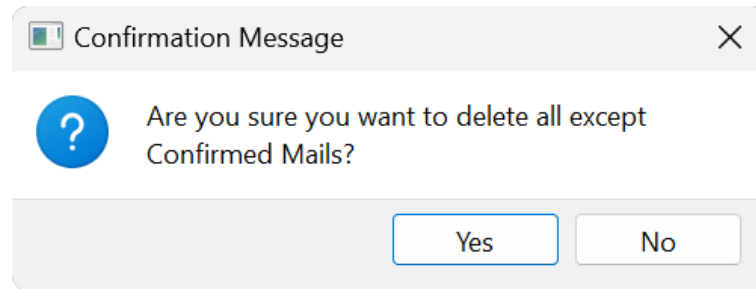


Figure 2.23: DEL Not Confirmed Message Box

If user clicks "*Yes*", information (reply) e-mails are deleted from inbox, and it creates another message box with the text "*Deleted successfully*", and the title "*Confirmed Info*", as shown in Figure 2.20.

This event will also result in "*All e-mails deleted except Confirmed Mails.*" being written in the log, as shown in Figure 2.24:

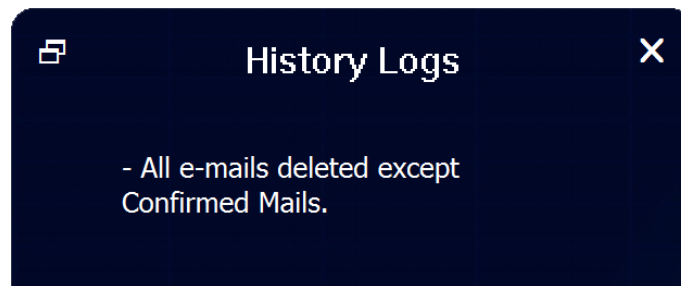


Figure 2.24: DEL Not Confirmed Log Record

If user clicks "*No*", the process is cancelled, and it creates another message box with the text "*Cancelled by user*", and the title "*Confirmed Info*", as shown in Figure 2.22.

2.3.3 Preferences

At the bottom of the Settings page there are two additional options that can be selected with a check box.

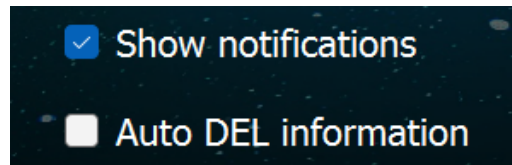


Figure 2.25: Preference Settings

Show Notifications

This check box, shown in Figure 2.25 is enabled by default. This means that the user will display all generated message boxes. If disabled, these message boxes will not be displayed by the user. This event will result in "*Notifications will not be displayed.*" being written in the log, as shown in Figure 2.26:

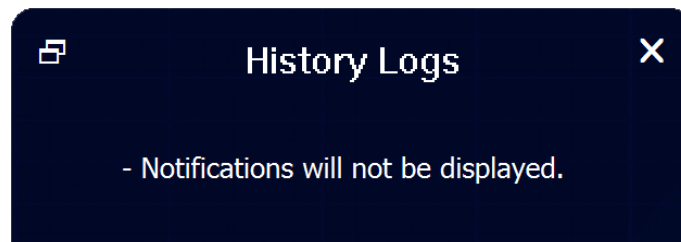


Figure 2.26: Notifications Off Log Record

If enabled again, the message boxes will be displayed again by the user. This event will result in "*Notifications will be displayed.*" being written in the log, as shown in Figure 2.27:

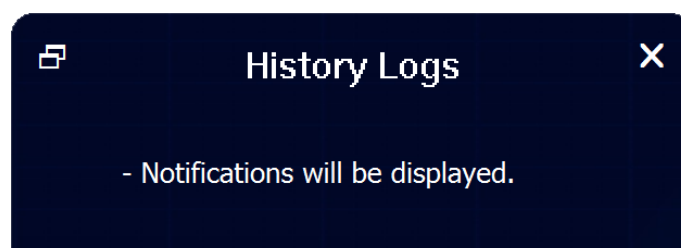


Figure 2.27: Notifications On Log Record

Auto DEL Information

The other check box, shown in Figure 2.25 is disabled by default. This function works similarly to the "*DEL information*" button. However, if enabled, the information (reply) e-mail is deleted after being read by Cmail.

This event will result in "*Information e-mails will be deleted after reading.*" being written in the log, as shown in Figure 2.28:

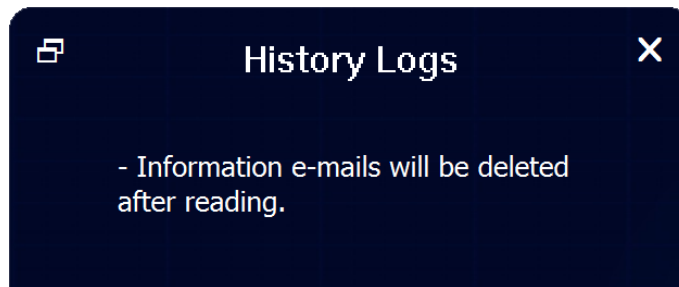


Figure 2.28: Auto DEL On Log Record

If disabled again, information (reply) e-mail is not deleted after it has been read by Cmail. This event will result in "*Information e-mails will not be deleted after reading.*" being written in the log, as shown in Figure 2.29:

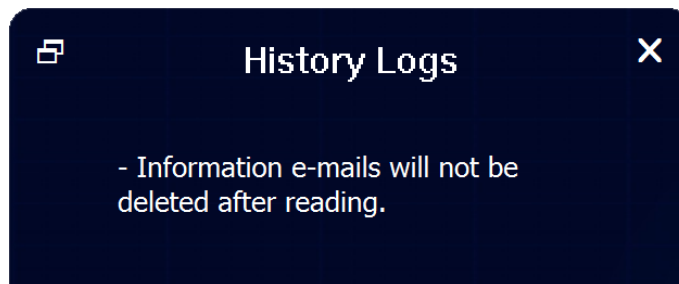


Figure 2.29: Auto DEL Off Log Record

2.4 Info Page

On this page, there is some information about Cmail and the creator of Cmail, as shown in Figure 2.30. The user can access the Cmail Guide, which is the GitHub page of Cmail. The user can also find the contact addresses of the creator of Cmail on this page.

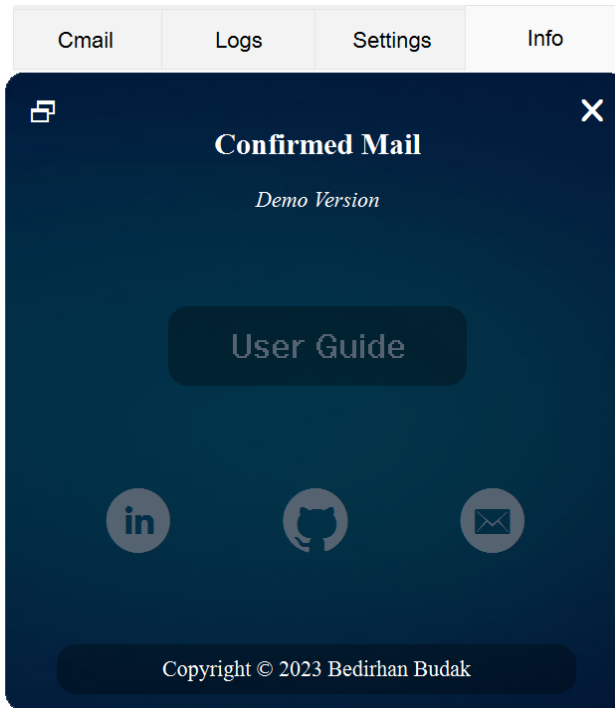


Figure 2.30: Info Page

2.5 Other Features

In this section, features of Cmail not previously mentioned will be explained.

2.5.1 Receiving an E-Mail

Cmail constantly checks the inbox of the e-mail address, and each time a new e-mail arrives, an event will occur based on the type of email.

New E-Mail

When user receives a new e-mail while Cmail is running, this will also result in "*You have a new e-mail!*" being written in the log, as shown in Figure 2.31:

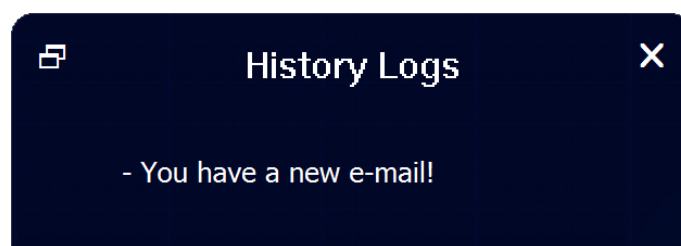


Figure 2.31: New E-Mail Log Record

Information E-Mail

If the response e-mail is replied by the sender, user will have an information e-mail, which will result in "*You have a new information e-mail.*" being written in the log, as shown in Figure 2.32:

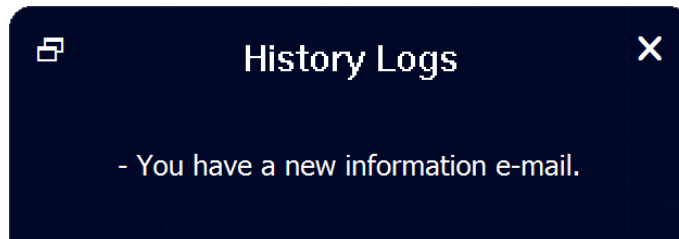


Figure 2.32: New Information E-Mail Log Record

Confirmed Mail

If an e-mail is confirmed, it creates a message box with the text "*You have a new Confirmed Mail!*", and the title "*Confirmed Mail*", as shown in Figure 2.33:

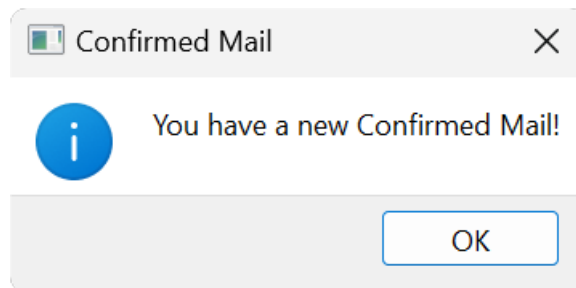


Figure 2.33: New Confirmed Mail Message Box

This event will also result in a green "*You have a new Confirmed Mail!*" to be written in the log, as shown in Figure 2.34:

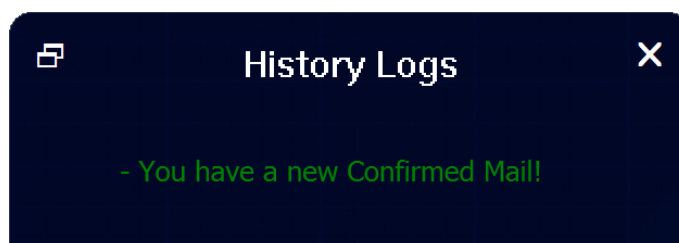


Figure 2.34: New Confirmed Mail Log Record

2.5.2 Tray Menu

The last feature is that Cmail has tray menu options, as shown in Figure 2.35:

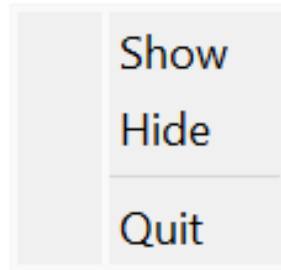


Figure 2.35: Tray Menu Options

User can display or hide the Cmail user interface or close the application completely.

2.5.3 Minimize & Quit Buttons

The user can also do these with the buttons available in each page of Cmail. Each page has a minimize button, and an exit button, as shown in Figure 2.36:



Figure 2.36: Minimize & Quit Buttons

Minimize Button

This button minimizes the application and is used to remove the application from the taskbar so that it runs in the background. If the user wants to restore the application, they can click on its icon in the tray menu. When the button is clicked, it creates a message box with the text "*Cmail is minimized.*", and the title "*Information*", as shown in Figure 2.37:

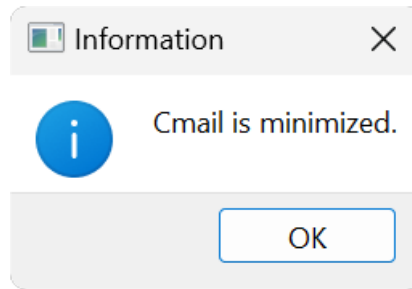


Figure 2.37: Minimized Message Box

This event will also result in a green "*Cmail is minimized.*" to be written in the log, as shown in Figure 2.38:

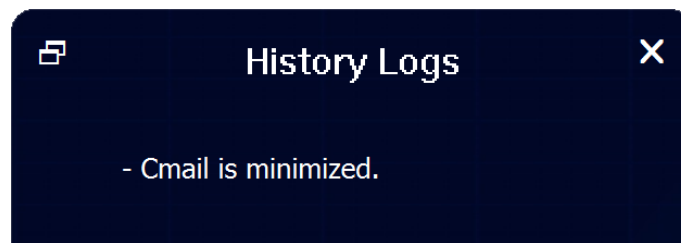


Figure 2.38: Minimized Log Record

Quit Button

This button allows to close the application. When this button is pressed, it creates a message box with the text "*Are you sure you want to quit Cmail?*", and the title "*Confirmation Message*" as shown in Figure 2.39:

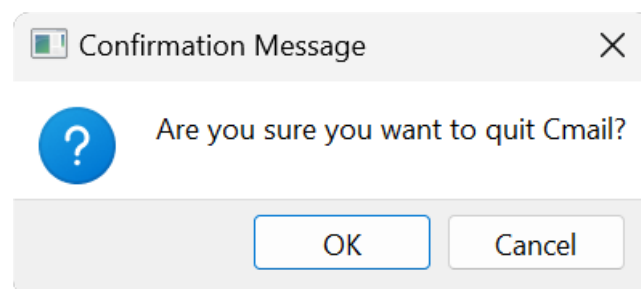


Figure 2.39: Quit Message Box

If the user clicks "*OK*", the application closes. If the user clicks "*Cancel*", this action is cancelled.

CHAPTER 3

Source Code of Confirmed Mail

This chapter describes the source code of the Confirmed Mail project in detail. The source code is published at: <https://github.com/bedirhanbudak/Cmail>

3.1 cmail.py

This file contains the basic code of the project running in the background. Processes such as receiving incoming e-mails, sending response e-mails, and checking incoming e-mails are performed with the code in this file.

3.1.1 Variable Definitions

The definitions of the global variables used in the "*cmail.py*" file are shown in Code 3.1:

```
user_email_address = str()
user_password = str()
confirmation_codes = dict()
logs = {"email": False, "info": False, "cmail": False, "del_info":
        False}
imap_protocol = str()
smtp_protocol = str()
smtp_port = int()
```

Code 3.1: Definitions of the Global Variables

"*user_email_address*" and "*user_password*" variables get their values from the line edits on the Cmail page, and similarly, "*smtp_protocol*", "*smtp_port*" get their values from the line edits on the Settings page. They will be defined in Code 3.20.

"*confirmation_codes*" dictionary holds the incoming e-mail IDs and the confirmation code sent to the e-mails belonging to these IDs. Therefore, the comparison between the confirmation code sent and the confirmation code received in reply is done through this variable.

"*logs*" dictionary is used to create log records. The keys in the dictionary check if some action has been performed, in this way they are used as flags. When an action is performed, their values change to True, and this change generates a signal that calls the functions that generate the log records.

3.1.2 start() Function

"*start()*" function is a function that starts Cmail and performs basic background processes. These processes are explained in the following subsections respectively.

Connecting to the Server

The function, defined in Code 3.2, connects to the e-mail server and is set to search for new e-mails in the inbox.

```
def start():
    time.sleep(2) # Delay to ensure connection

    # Connect and login to the e-mail server
    email = imaplib.IMAP4_SSL(imap_protocol)
    email.login(user_email_address, user_password)

    time.sleep(2) # Delay to ensure connection

    # Select the inbox and search for new e-mails
    email.select("inbox")
    status, messages = email.search(None, "RECENT")
    messages = messages[0].split() # Get the list of new e-mail IDs
```

Code 3.2: Connection and Searching

Parsing New E-Mail

The function starts the iteration for a new incoming e-mail in Code 3.3. When a new e-mail arrives, it parses its information and proceeds with the following steps.

```
for email_id in messages:
    time.sleep(1) # Delay to ensure connection

    # Get the e-mail data
    status, email_data = email.fetch(email_id, "(RFC822)")

    # Assign the e-mail data to email_message
    email_message = email.message_from_bytes(email_data[0][1])

    # Extract the sender's e-mail address information
    sender_email = email_message["From"]
    sender_subject = email_message["Subject"]
```

Code 3.3: Parsing New E-mails

Confirmation for E-Mail

If the new incoming e-mail is an information (reply) e-mail titled "*Re: Confirmed Mail*" (the "*Re*" part may change depending on language), the condition works:

```
if ": Confirmed Mail" in sender_subject:
    logs["info"] = True # Log record for new information e-mail
    email_content = email_data[0][1].decode("utf-8")

    # Search for the confirmation code present if in the content
    for confirmed_key, confirmed_code in confirmation_codes.items():
        if confirmed_code in email_content: # If code is matched
            email.store(confirmed_key, '+FLAGS', '\\Flagged')
            logs["cmail"] = True # Log record for new Confirmed Mail

        if logs["del_info"]: # If "Auto DEL" is checked
            email.store(email_id, '+FLAGS', '\\Deleted')
            logs["info"] = False # Log disabled for new info
```

Code 3.4: Confirming an E-Mail

This condition, defined in Code 3.4, checks the confirmation code inside the e-mail. If the code sent is found in the reply e-mail and matches the code in the value of the key in the dictionary that holds the ID of the e-mail, that e-mail is marked (flagged). Additionally, if the user has enabled "*Auto DEL information*" in the settings, this information e-mail is marked for deletion.

Sending Response to Incoming E-Mail

If the new incoming e-mail is an information (reply) e-mail titled "*Re: Confirmed Mail*" (the "*Re*" part may change depending on language), the condition, defined under this subsection, works to send a response to a new incoming e-mail.

```
else:
    logs["email"] = True # Log record for new e-mail

    # Generate a confirmation code and add it to the dictionary
    confirmation_code = authenticator.create_code()
    confirmation_codes.update({email_id: confirmation_code})
```

Code 3.5: Creating Confirmation Code

Firstly, in Code 3.5, the value of "*logs[\"email\"]*" key is set to True to generate a log record that a new e-mail has arrived. It will be defined in Subsection 3.4.3.

After that, a confirmation code is generated and stored in the "*confirmation_codes*" dictionary along with the ID of the generated e-mail. The function used to generate the code will be described in Section 3.2. The condition then proceeds as follows.

```
# Search for name and e-mail pattern in the sender_email
match = re.search(r"^(.*?)\s*<([\w\.-]+)@([\w\.-]+)", sender_email)

if match:
    # Extract the captured username
    username = match.group(1) or match.group(2)

    # Decode the username if it is encoded
    decoded_parts = decode_header(username)
    decoded_username = ""
```

```
for part, encoding in decoded_parts:
    if isinstance(part, bytes):
        decoded_username += part.decode(encoding or "utf-8",
                                         errors="ignore")
    else:
        decoded_username += part

# Remove leading and trailing whitespace from the username
username = decoded_username.strip()

else:
    username = "Unknown"
```

Code 3.6: Extracting the Username of Sender

Secondly, in Code 3.6, the sender's username is decoded to "UTF-8" for inclusion in the response e-mail. To do this, the function uses the `re.search()` function from the `re` module to search for a pattern in the `sender_email` string.

The `decode_header()` function is used to decode any encoded parts in the username. After that, the following lines iterate over the `decoded_parts` list and decode each part if it is encoded and assign to the `username` variable. If no match was found in the initial `re.search()` step, the `username` is set to `Unknown`.

```
# Extract the plain text part of the e-mail
plain_text_part = None
for part in email_message.walk():
    if part.get_content_type() == 'text/plain':
        plain_text_part = part
        break

# Check if a plain text part was found and decode it
if plain_text_part:
    original_content = plain_text_part.get_payload(decode=True).
        decode("utf-8", errors="ignore")
else:
    original_content = "No plain text content found in the e-mail."
```

Code 3.7: Extracting the Content of Incoming E-Mail

Thirdly, in Code 3.7, the content of the incoming e-mail is extracted in plain text and decoded to "UTF-8" for inclusion in the response e-mail. If the content of the incoming e-mail is not found, the content is set to "No plain text content found in the e-mail."

```
# Response message content
response_message = MIMEText("Hi " + username + ",\n\n"
    "This e-mail address is secured by Confirmed Mail system\n"
    "designed to prevent phishing attacks. "\n\n"
    "Read more about Cmail: github.com/bedirhanbudak/Cmail\n\n"
    "(!) To confirm your identity, please reply to this e-mail\n"
    "with this code: " + confirmation_code + "\n"
    "(!) If you are not the sender of the e-mail shown below,\n"
    "DO NOT reply to this e-mail!\n\n"
    "Feel free to add any additional information.\n\n"
    "Sincerely yours,\n\n"
    "Confirmed Mail\n\n"
    "-----\n\n"
    "The content of your e-mail to be confirmed:\n\n" +
    original_content, _charset="utf-8")

# Settings for response e-Mail
response_message["Subject"] = "Confirmed Mail"
response_message["From"] = user_email_address
response_message["To"] = sender_email

# Send the response e-mail
server = smtplib.SMTP(smtp_protocol, smtp_port)
server.starttls()
server.login(user_email_address, user_password)
server.send_message(response_message)
server.quit()
```

Code 3.8: Creating Response Message and Sending E-Mail

Finally, in Code 3.8, the response e-mail is set. This e-mail contains the sender's name, the generated unique confirmation code, and the plain text of the incoming e-mail. After assigning the relevant values to the variables that will be used to send the response e-mail, the e-mail is sent by connecting to the SMTP server.

3.1.3 del_info() Function

This function is used to delete information (reply) e-mails that are redundant for the user in the inbox.

```
def del_info():  
    # Create an IMAP4_SSL object and login to the e-mail server  
    email = imaplib.IMAP4_SSL(imap_protocol)  
    email.login(user_email_address, user_password)  
  
    # Select the "inbox" folder  
    email.select("inbox")  
  
    # Search for e-mails with the subject ": Confirmed Mail"  
    status, emails = email.search(None,  
                                  'Subject ": Confirmed Mail"')  
  
    # Get the list of e-mail IDs from the search results  
    email_ids = emails[0].split()  
  
    # Iterate over the e-mail IDs and mark them for deletion  
    for email_id in email_ids:  
        email.store(email_id, '+FLAGS', '\\Deleted')  
  
    # Logout from the e-mail server  
    email.logout()
```

Code 3.9: Deleting Information E-Mails

When this function, defined in Code 3.9, is called, it logs in to the entered e-mail address again and searches for the information (reply) e-mails titled "*Re: Confirmed Mail*" (the "*Re*" part may change depending on language) in the inbox and marks them all as deleted.

After the deletion process is finished, logged out from the e-mail server.

3.1.4 del_email() Function

This function deletes all mails except confirmed (marked) ones in the inbox to make it easier for the user.

```
def del_email():  
    # Create an IMAP4_SSL object and login to the e-mail server  
    email = imaplib.IMAP4_SSL(imap_protocol)  
    email.login(user_email_address, user_password)  
  
    # Select the "inbox" folder  
    email.select("inbox")  
  
    # Search for emails that are marked as "UNFLAGGED"  
    status, emails = email.search(None, "UNFLAGGED")  
  
    # Get the list of e-mail IDs from the search results  
    email_ids = emails[0].split()  
  
    # Iterate over the email IDs and mark them for deletion  
    for email_id in email_ids:  
        email.store(email_id, '+FLAGS', '\\Deleted')  
  
    # Logout from the e-mail server  
    email.logout()
```

Code 3.10: Deleting Not Confirmed E-Mails

When this function, defined in Code 3.10, is called, it logs in to the entered e-mail address again and searches for the not confirmed e-mails that are not flagged (marked) in the inbox and marks them all as deleted.

After the deletion process is finished, logged out from the e-mail server.

3.2 authenticator.py

Confirmation codes are generated in this file. This feature is written using ready-made libraries. It is written in a separate file so that the generated code can be improved and made more reliable in the future. The complete code in the file is shown in Code 3.11:

```
import pyotp

# Create a secret key
secret = pyotp.random_base32()

# Generate a TOTP object
totp = pyotp.TOTP(secret)

def create_code():
    # Get the current TOTP code
    totp_code = totp.now()

    # Return the generated TOTP code
    return totp_code
```

Code 3.11: Generating Confirmation Codes

"*pyotp*" library is used to generate a "*Time-Based One-Time Password (TOTP)*" using the Python library. This code works as follows:

Firstly, a random secret key is generated using the "*random_base32()*" function from the "*pyotp*" library. This secret key is used by the TOTP algorithm. Next, a TOTP object is created using the TOTP class from the "*pyotp*" library. The created TOTP object generates one-time passwords using the secret key. When the "*create_code()*" function is called, the current TOTP code "*totp.now()*" is obtained and returned.

To sum up, this code generates a different secret key each time it is run and returns the current TOTP code whenever "*create_code()*" is called.

3.3 design.py

"Qt Designer" application is used to design the interface of the Confirmed Mail project. Qt Designer includes "Qt Widgets" for designing and building graphical user interfaces (GUIs). It is easy to use and works with drag-and-drop.

3.3.1 Form & Tab Widget

Cmail has a "QWidget" named "Form_cmail", which is the main form, and a "QTabWidget" is placed inside the main form, as shown in Figure 3.1:



Figure 3.1: From and Tab Widget in Qt Designer

Design of Cmail is based on this tab widget. This widget "tabWidget_cmail" is set to a fixed size of 320 x 360. It has four "QWidget" tabs named "tab_cmail", "tab_logs", "tab_settings", "tab_info". The tools on each page are placed on a grid layout.

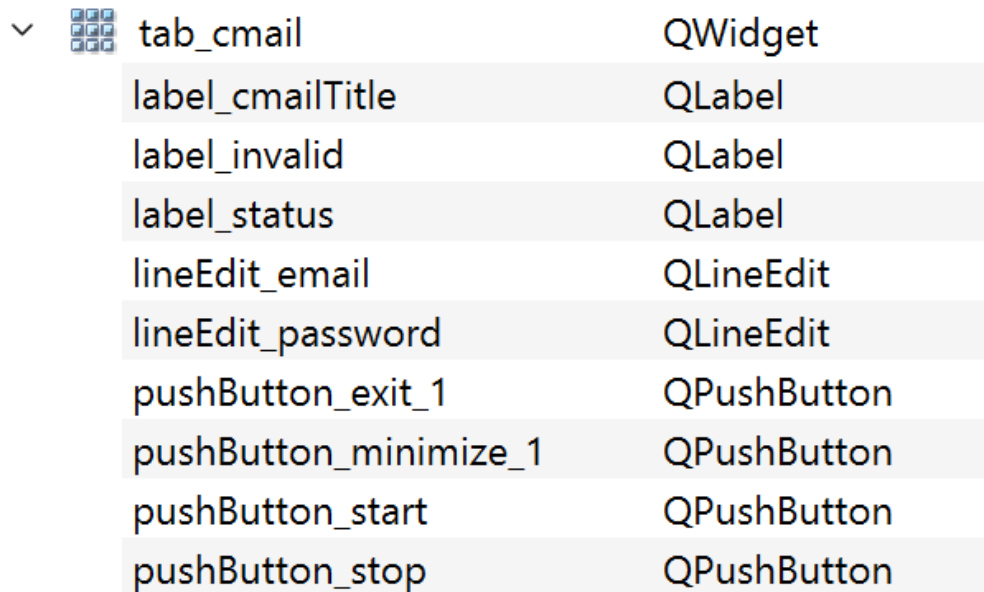
3.3.2 Minimize & Quit Buttons

All pages have "pushButton_exit" and "pushButton_minimize" buttons. On each page, the number at the end of their name increases, such as "pushButton_exit_1" and "pushButton_exit_2". They are introduced under this subsection so that they are not mentioned again each time.

Webdings characters are used as the character set of for the push buttons. Hovering the mouse over the buttons changes their colors. Clicking on them changes their colors again, and the symbol on them scrolls down to the right. These buttons are placed at the top left and top right of the tab widget so that they are the same on each page.

3.3.3 Cmail Tab

The first tab, which is "*Cmail Tab*", has three "*QLabel*", two "*QLineEdit*", and four "*QPushButton*", as shown in Figure 3.2:




▼  tab_cmail	QWidget
label_cmailTitle	QLabel
label_invalid	QLabel
label_status	QLabel
lineEdit_email	QLineEdit
lineEdit_password	QLineEdit
pushButton_exit_1	QPushButton
pushButton_minimize_1	QPushButton
pushButton_start	QPushButton
pushButton_stop	QPushButton

Figure 3.2: Cmail Tab in Qt Designer

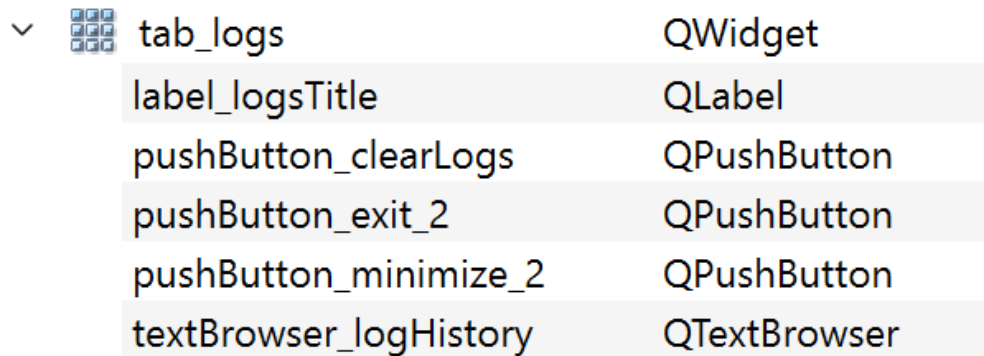
The line edits "*lineEdit_email*" and "*lineEdit_password*" are for entering information by the user. Therefore, they have placeholder text in the form of "*E-mail address*" and "*Password*". Furthermore, the echo mode of "*lineEdit_password*" is of password type to hide the password text when the user enters it. They are placed in the center of the grid layout.

The label "*label_cmailTitle*" is the title of this tab, "*Confirmed Mail*". It is decorated with a bottom border. Furthermore, "*label_invalid*" and "*label_status*" are labels used to inform the user and are colored differently. They are placed at the top, center, and bottom of the grid layout.

Push buttons "*pushButton_start*" and "*pushButton_stop*" are decorated to have the effect. The corners of the buttons are rounded. Hovering the mouse over the buttons changes their colors. Clicking on them changes their colors again, and the texts on them scrolls down to the right. These buttons are placed below the line edits.

3.3.4 Logs Tab

The second tab, which is "*Logs Tab*", has a "*QLabel*", three "*QPushButton*", and a "*QTextBrowser*", as shown in Figure 3.3:




▼  tab_logs	QWidget
label_logsTitle	QLabel
pushButton_clearLogs	QPushButton
pushButton_exit_2	QPushButton
pushButton_minimize_2	QPushButton
textBrowser_logHistory	QTextBrowser

Figure 3.3: Logs Tab in Qt Designer

The label "*label_logsTitle*" is the title of this tab, "*History Logs*". It is placed at the top of the grid layout.

The text browser "*QTextBrowser*" is set to a fixed size of 220 x 200 and placed in the center of the grid layout.

The push button "*pushButton_clearLogs*" is decorated to have this effect. Hovering the mouse over the buttons changes their colors, and clicking on them changes their colors again. Furthermore, the background is removed. These buttons are placed at the bottom of the grid layout.

3.3.5 Settings Tab

The third tab, which is "*Settings Tab*", has two "*QCheckBox*", two "*QGroupBox*", four "*QLabel*", four "*QPushButton*", and three "*QLineEdit*", as shown in Figure 3.4:




▼ 	tab_settings	QWidget
	checkBox_autoDEL	QCheckBox
	checkBox_notifications	QCheckBox
▼ 	groupBox_cleaning	QGroupBox
	label_warning	QLabel
	pushButton_allDEL	QPushButton
	pushButton_infoDEL	QPushButton
▼ 	groupBox_service	QGroupBox
	label_imapServer	QLabel
	label_smtpPort	QLabel
	label_smtpServer	QLabel
	lineEdit_imapServer	QLineEdit
	lineEdit_smtpPort	QLineEdit
	lineEdit_smtpServer	QLineEdit
	pushButton_exit_3	QPushButton
	pushButton_minimize_3	QPushButton

Figure 3.4: Settings Tab in Qt Designer

The group box `"groupBox_service"` has six tools `"label_imapServer"`, `"lineEdit_imapServer"`, `"label_smtpServer"`, `"lineEdit_smtpServer"`, `"label_smtpPort"`, and `"lineEdit_smtpPort"`. These are placed at the top of the grid layout in the group box. The line edits have rounded corners, and each has placeholder text such as `"imap.example.com"`, `"smtp.example.com"`, and `"xxx"`.

The group box `"groupBox_cleaning"` has three tools `"label_warning"`, `"pushButton_allDel"`, and `"pushButton_infoDel"`. These are placed in the center of the grid layout in the group box. The text inside the label is colored in red.

Checkboxes `"checkBox_autoDEL"` and `"checkBox_notifications"` are placed at the bottom of the grid layout. Additionally, the `"checkBox_notifications"` is checked as default.

3.3.6 Info Tab

The fourth tab, which is "Info Tab", has a "QHBoxLayout", seven "QPushButton", and a "QLabel", as shown in Figure 3.5:

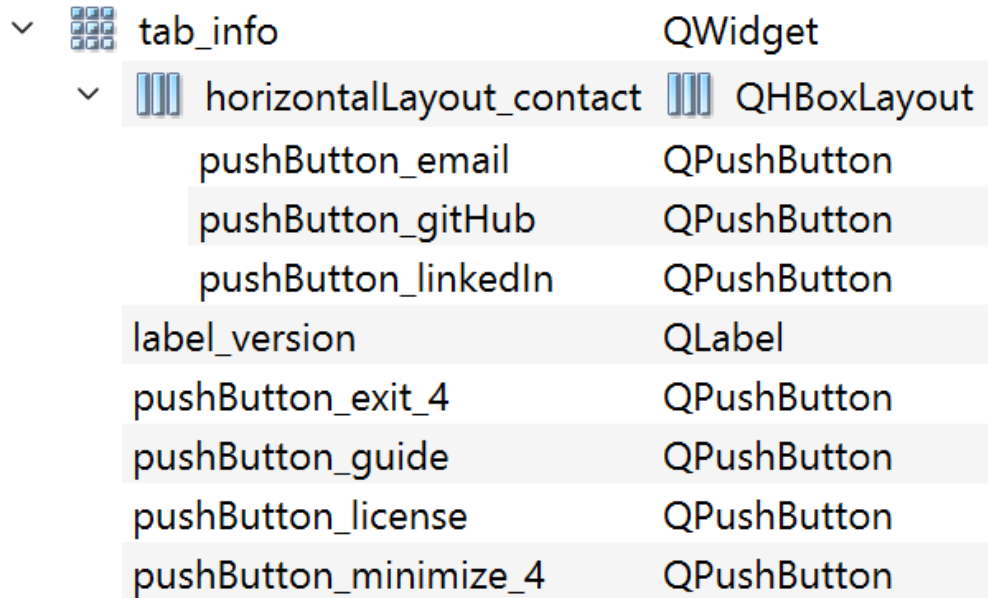


Figure 3.5: Info Tab in Qt Designer

The label "*label_version*" is placed at the top of the grid layout. It contains the name and version of the application.

The push button "*pushButton_guide*" is decorated to have the effect. Hovering the mouse over the button changes its color. Clicking on it changes its color again, and the text on it scrolls down to the right. Furthermore, the background is shaded. This button is placed at the top center of the grid layout.

The horizontal layout "*horizontalLayout_contact*" has three push buttons "*pushButton_email*", "*pushButton_gitHub*", and "*pushButton_linkedin*". These buttons are lined up side by side and Social Media Circled characters are used as the character set of the buttons. This horizontal layout is placed at the bottom center of the grid layout with these buttons.

The push button "*pushButton_license*" is decorated to have the effect. Hovering the mouse over the button changes its color. Clicking on it changes its color again, and the text on it scrolls down to the right. Furthermore, the background is shaded. This button is placed at the bottom of the grid layout.

3.4 main.py

This is the main file of the Cmail application. It inherits from "*cmail.py*" and "*design.py*" to call their functions. The actions that the user performs in the Cmail interface as well as the various background threads are defined in this file.

3.4.1 Cmail Class

The actions that the user interacts with the interface are defined in this class.

```
class Cmail(QtWidgets.QMainWindow, Ui_Form_cmail):
    def __init__(self):
        super().__init__() # Call the parent class constructor
        QThread.__init__(self) # Initialize the QThread base class
        self.uiDesign = Ui_Form_cmail() # Instance of Ui_Form_cmail
        self.uiDesign.setupUi(self) # Set up the UI design
        self.thread = LoopThread() # Instance of LoopThread
```

Code 3.12: Cmail Class Definition

"Cmail class" inherits from "*Ui_Form_cmail*" in the "*design.py*" file, as defined in Code 3.12, and all connections with this file are provided in this class.

3.4.2 Connections

This section describes the connections between actions and functions.

Signal Connections

A few signals are emitted from the LoopThread class when conditions are met.

```
self.thread.email_signal.connect(self.logs_email)
self.thread.info_signal.connect(self.logs_info)
self.thread.cmail_signal.connect(self.logs_cmail)
self.thread.error_signal.connect(self.login_error)
```

Code 3.13: Signal Connections

In Code 3.13, it is defined to connect the emitted signals to the corresponding functions when the "*run()*" function in the LoopThread class is running.

Button Connections

This subsection describes which buttons are connected to which functions. The functions will be defined in Subsection 3.4.3.

```
self.uiDesign.pushButton_start.clicked.connect(self.clicked_on_start)
self.uiDesign.pushButton_stop.clicked.connect(self.clicked_on_stop)
```

Code 3.14: Button Connections on Cmail Page

On Cmail page, "*pushButton_start*" and "*pushButton_stop*" buttons are connected to a function, as defined in Code 3.14.

```
self.uiDesign.pushButton_clearLogs.clicked.connect(self.uiDesign.
    textBrowser_logHistory.clear)
```

Code 3.15: Button Connections on Logs Page

On Logs page, "*pushButton_clearLogs*" button is connected to the "*textBrowser_logHistory.clear*" function, as defined in Code 3.15, which is used to clear the logs.

```
self.uiDesign.pushButton_infoDEL.clicked.connect(self.del_info)
self.uiDesign.pushButton_allDEL.clicked.connect(self.del_email)
self.uiDesign.checkBox_notifications.stateChanged.connect(self.
    check_notification)
self.uiDesign.checkBox_autoDEL.stateChanged.connect(self.check_
    deleteInfo)
```

Code 3.16: Button Connections on Setting Page

On Settings page, "*pushButton_infoDEL*" and "*pushButton_allDEL*" buttons are connected to a function. Furthermore, "*checkBox_notifications*" and "*checkBox_autoDEL*" checkboxes are connected to a function that will run when they are checked or unchecked, as defined in Code 3.16.

```
self.uiDesign.pushButton_guide.clicked.connect(self.link_guide)
self.uiDesign.pushButton_license.clicked.connect(self.link_license)
```

Code 3.17: Button Connections on Info Page

On Info page, there are several buttons that work similarly and the functions that some of them are connected to are defined in Code 3.17.

```
self.uiDesign.pushButton_minimize.clicked.connect(self.background_
    clicked)
self.uiDesign.pushButton_exit.clicked.connect(self.exit_app)
```

Code 3.18: Button Connections in Minimize & Quit Buttons

Each "*pushButton_minimize*" and "*pushButton_exit*" button is connected to the same functions, "*background_clicked*" and "*exit_app*", as defined in Code 3.18.

Tray Menu

Tray menu processes are defined in Code 3.19:

```
self.tray_icon = QtWidgets.QSystemTrayIcon(self)
self.tray_icon.setIcon(QIcon("images/logo.png"))
self.tray_icon.setToolTip("Cmail is running in background.")
self.tray_icon.show()
self.tray_icon.activated.connect(self.on_tray_icon_clicked)
show_action = QtWidgets.QAction("Show", self)
show_action.triggered.connect(self.showNormal)
minimize_action = QtWidgets.QAction("Hide", self)
minimize_action.triggered.connect(self.hide)
quit_action = QtWidgets.QAction("Quit", self)
quit_action.triggered.connect(QtWidgets.qApp.quit)
tray_menu = QtWidgets.QMenu()
tray_menu.addAction(show_action)
tray_menu.addAction(minimize_action)
tray_menu.addSeparator()
tray_menu.addAction(quit_action)
self.tray_icon.setContextMenu(tray_menu)
```

Code 3.19: Tray Menu

Firstly, the tray icon and tooltip are set so that Cmail's icon appears in the tray menu. After that, when the tray icon is clicked with the left mouse, the action is connected to the "*on_tray_icon_clicked*" function. Right-clicking on the tray icon opens a menu with the actions "*Show*", "*Hide*", and "*Quit*".

3.4.3 Functions

This section defines all functions together with the functions mentioned above.

`clicked_on_start()`

This function is called when the user clicks on the "*pushButton_start*" button.

```
def clicked_on_start(self):
    if self.check_format() and not
        self.uiDesign.lineEdit_password.text().strip() == "":
        cmail.user_email_address = self.uiDesign.lineEdit_email.text()
        cmail.user_password = self.uiDesign.lineEdit_password.text()
        cmail.imap_protocol = self.uiDesign.lineEdit_imapServer.text()
        cmail.smtp_protocol = self.uiDesign.lineEdit_smtpServer.text()
        cmail.smtp_port = self.uiDesign.lineEdit_smtpPort.text()
        self.uiDesign.label_invalid.clear()
        self.thread.start()
        self.uiDesign.label_status.setText("Cmail is running")
        self.uiDesign.label_status.setStyleSheet("color: green;")
        self.uiDesign.pushButton_stop.setEnabled(True)
        self.uiDesign.pushButton_start.setEnabled(False)
        self.uiDesign.pushButton_infoDEL.setEnabled(True)
        self.uiDesign.pushButton_allDEL.setEnabled(True)
        self.uiDesign.lineEdit_email.setEnabled(False)
        self.uiDesign.lineEdit_password.setEnabled(False)
        self.uiDesign.lineEdit_imapServer.setEnabled(False)
        self.uiDesign.lineEdit_smtpServer.setEnabled(False)
        self.uiDesign.lineEdit_smtpPort.setEnabled(False)
        if self.uiDesign.checkBox_notifications.isChecked():
            self.message_box_information("Confirmed Info",
                                         "Cmail has been started.")
            self.add_log("Cmail has been started.")
    else:
        self.uiDesign.label_invalid.setText("Invalid format!")
        self.message_box_information("Error", "Invalid e-mail address!")
        self.add_log("Invalid e-mail address entered.")
```

Code 3.20: `clicked_on_start()`

If the `"check_format"` variable is `True` (to be defined in Code 3.22), and if the `"lineEdit_password"` field is not empty, the function assigns the values of the line edits on the Cmail page to variables in `"cmail.py"`. After that, the `"start()"` function of the `LoopThread` class (to be defined in Subsection 3.4.4) is called.

Meanwhile, the text of the status labels and the accessibility of some buttons change to inform the user. A message box is also created with the text *"Cmail has been started."* and the title *"Confirmed Info"* if `"checkBox_notifications"` is checked, and it also results in *"Cmail has been started."* to be written in the log.

If `"check_format"` is `False` or the `"lineEdit_password"` is empty, the text of the `"label_invalid"` is changed to *"Invalid format!"* and a message box is created with the text *"Invalid e-mail address or password!"* and the title *"Error"*, then it also results in *"Invalid e-mail address or password entered."* to be written in the log.

`clicked_on_stop()`

This function is called when the user clicks on the `"pushButton_stop"` button.

```
def clicked_on_stop(self):
    self.thread.stop()
    self.uiDesign.label_status.setText("Cmail has stopped")
    self.uiDesign.label_status.setStyleSheet("color: red;")
    self.uiDesign.pushButton_start.setEnabled(True)
    self.uiDesign.pushButton_stop.setEnabled(False)
    self.uiDesign.pushButton_infoDEL.setEnabled(False)
    self.uiDesign.pushButton_allDEL.setEnabled(False)
    self.uiDesign.lineEdit_email.setEnabled(True)
    self.uiDesign.lineEdit_password.setEnabled(True)
    self.uiDesign.lineEdit_imapServer.setEnabled(True)
    self.uiDesign.lineEdit_smtpServer.setEnabled(True)
    self.uiDesign.lineEdit_smtpPort.setEnabled(True)
    if self.uiDesign.checkBox_notifications.isChecked():
        self.message_box_information("Confirmed Info", "Cmail has
            been stopped.")
    self.add_log("Cmail has been stopped.")
    cmail.confirmation_codes.clear()
    cmail.logs = {key: False for key in cmail.logs}
```

Code 3.21: `clicked_on_stop()`

The function is called when the user clicks on the "*pushButton_stop*" button. At the beginning of the function, the "*stop()*" function of the *LoopThread* class is called, which means *Cmail* is stopped.

Meanwhile, the text of the status labels and the accessibility of some buttons change to inform the user. A message box is also created with the text "*Cmail has been stopped.*" and the title "*Confirmed Info*" if "*checkBox_notifications*" is checked, and it also results in "*Cmail has been stopped.*" to be written in the log.

At the end of the function, the dictionary holding the confirmation codes is cleared, which means that previous e-mails can no longer be confirmed. Furthermore, the variables holding the log flags are set to default (*False*) to avoid issues due to interruption.

check_format()

The function checks if the format of the entered e-mail address is valid, as defined in Code 3.22:

```
def check_format(self):
    text = self.uiDesign.lineEdit_email.text()
    email_regex = re.compile(r'[\w\.-]+@[\w\.-]+\.[\w\.-]')
    if email_regex.match(text):
        return True
    else:
        return False
```

Code 3.22: check_format()

The function is called inside the "*clicked_on_start*" function, defined in Code 3.20. It assigns the e-mail address entered in the line edit to the "*text*" variable. Then, it defines a regular expression pattern to match e-mail format as "*email_regex*". This regex keeps strings before and after the character "@" and before and after the character ".". If these two variables match, the function returns *True*. If not, it returns *False*. Therefore, the email format should be "*example@example.example*".

login_error()

The function, defined in Code 3.23, is called when the entered e-mail address and password do not match. Hence, there is a login error.

```
def login_error(self):
    if self.uiDesign.checkBox_notifications.isChecked():
        self.message_box_information("Error", "The e-mail address or
            password is incorrect!")
    self.add_log("Failed to login!")
    self.uiDesign.label_invalid.setText("Login failed!")
    self.clicked_on_stop()
```

Code 3.23: login_error()

When "*error_signal*" is emitted from the LoopThread class, it calls the function with the connection defined in Code 3.13. In this function, a message box is created with the text "*The e-mail address or password is incorrect!*" and the title "*Error*" if "*checkBox_notifications*" is checked, and it also results in "*Failed to login!*" to be written in the log. Additionally, the text of the "*label_invalid*" label is changed to "*Login Failed!*". After all, the "*clicked_on_stop*" function that is defined in Code 3.21 is called, and Cmail is stopped.

logs_email()

The function, defined in Code 3.24, is called when getting a new e-mail.

```
def logs_email(self):
    cmail.logs["email"] = False # Set to default value to reuse
    self.add_log("You have a new e-mail!")
```

Code 3.24: logs_email()

When the user receives a new e-mail, a signal is emitted from the LoopThread class, and this function is called by the connection that is described in Code 3.13. In this function, the value of the "*logs['email']*" key is assigned to False again, and "*You have a new e-mail!*" is written to the log.

logs_info()

The function, defined in Code 3.25, is called when getting a new information (reply) e-mail.

```
def logs_info(self):  
    cmail.logs["info"] = False # Set to default value to reuse  
    self.add_log("You have a new information e-mail.")
```

Code 3.25: logs_info()

When the user receives a new information e-mail, a signal is emitted from the LoopThread class, and this function is called by the connection that is described in Code 3.13. In this function, the value of the "logs["info"]" key is assigned to False again, and "You have a new information e-mail!" is written to the log.

logs_cmail()

The function, defined in Code 3.26, is called when getting a new Confirmed Mail.

```
def logs_cmail(self):  
    cmail.logs["cmail"] = False # Set to default value to reuse  
    self.uiDesign.textBrowser_logHistory.setTextColor(QColor  
        ('green'))  
    self.add_log("You have a new Confirmed Mail!")  
    self.uiDesign.textBrowser_logHistory.setTextColor(QColor  
        ('white'))  
    if self.uiDesign.checkBox_notifications.isChecked():  
        self.message_box_information("Confirmed Mail", "You have a  
            new Confirmed Mail!")
```

Code 3.26: logs_cmail()

When the user receives a new Confirmed Mail, a signal is emitted from the LoopThread class, and this function is called by the connection that is described in Code 3.13. In this function, the value of the "logs["cmail"]" key is assigned to False again, and "You have a new information e-mail!" is written in green in the log. Additionally, it also creates a message box with the text "You have a new Confirmed Mail." and the title "Confirmed Mail" if "checkBox_notifications" is checked.

del_info()

The function, defined in Code 3.27, is used to delete information e-mails.

```
def del_info(self):  
    self.message_box_question("Are you sure you want to delete  
        information e-mails?", self.thread.delete_info,  
        self.uiDesign.pushButton_infoDEL.setEnabled,  
        "Information e-mails deleted.")
```

Code 3.27: del_info()

This function is called when the user clicks on the button "*pushButton_infoDEL*" button while Cmail is running. It also creates a question message box with the text "*Are you sure you want to delete information e-mails?*". If user clicks on "Yes", the function "*self.thread.delete_info*" is called and the button "*pushButton_infoDEL*" is set to disabled. It also results in "*Information e-mails deleted.*" to be written in the log. If user clicks on "No", the process is canceled.

del_email()

The function, defined in Code 3.40, is used to delete not confirmed e-mails.

```
def del_email(self):  
    self.message_box_question("Are you sure you want to delete all  
        except Confirmed Mails?", self.thread.delete_email,  
        self.uiDesign.pushButton_allDEL.setEnabled,  
        "All e-mails deleted except Confirmed Mails.")
```

Code 3.28: del_email()

This function is called when the user clicks on the "*pushButton_allDEL*" button while Cmail is running. It also creates a question message box with the text "*Are you sure you want to delete all except Confirmed Mails?*". If user clicks on "Yes", the function "*self.thread.delete_email*" is called and the button "*pushButton_allDEL*" is set to disabled. It also results in "*All e-mails deleted except Confirmed Mails.*" to be written in the log. If user clicks on "No", the process is canceled.

Checkboxes

There are two checkboxes in the Settings page, and their actions are defined inside the functions Code 3.29 and Code 3.30:

```
def check_notification(self):
    if self.uiDesign.checkBox_notifications.isChecked():
        self.add_log("Notifications will be displayed.")
    if not self.uiDesign.checkBox_notifications.isChecked():
        self.add_log("Notifications will not be displayed.")
```

Code 3.29: check_notification()

The function defined in Code 3.29 only creates a log when this checkbox is checked or unchecked. Because when creating a message box anywhere, a condition is defined according to whether the checkbox "*checkBox_notifications*" is checked or unchecked.

```
def check_deleteInfo(self):
    if self.uiDesign.checkBox_autoDEL.isChecked():
        cmail.logs["del_info"] = True # Auto delete on
        self.add_log("Information e-mails will be deleted after
                      reading.")
    if not self.uiDesign.checkBox_autoDEL.isChecked():
        cmail.logs["del_info"] = False # Auto delete off
        self.add_log("Information e-mails will not be deleted after
                      reading.")
```

Code 3.30: check_deleteInfo()

The function defined in Code 3.30 generates a log record and changes the value of the "*logs*['*del_info*']" key according to whether the checkbox "*checkBox_autoDEL*" is checked or unchecked. If this value is True, the condition defined in Code 3.4 is satisfied.

Information Buttons

There are five push buttons on the Settings page, and the functions called by these buttons are defined in Code 3.31:

```
def link_guide(self): # Cmail GitHub Page
    QDesktopServices.openUrl(QUrl(
        'https://github.com/bedirhanbudak/Cmail'))

def link_license(self): # License message box
    self.message_box_information("License", "Confirmed Mail Demo
        Version\n\n" + "Copyright © 2023 Bedirhan Budak\n" +
        "under GNU GPL v3 license")
```

Code 3.31: Information Buttons

The function "*link_guide*" redirects the user to Cmail's GitHub page. Similarly defined but not shown here are the "*contact_linkedin*", "*contact_github*", and "*contact_email*" functions. These redirect to the LinkedIn, GitHub, and email addresses of Cmail's creator. The last function creates a message box with the text "*Confirmed Mail / Demo Version / Copyright © 2023 Bedirhan Budak / under GNU GPL v3 license*" and the title "*License*".

Minimize & Quit Buttons

The actions of the Minimize & Quit buttons are defined in Code 3.32:

```
def background_clicked(self):
    self.message_box_information("Information", "Cmail is minimized.")
    self.add_log("Cmail is minimized.")
    self.hide() # Hide the interface

def on_tray_icon_clicked(self, reason):
    if reason == QtWidgets.QSystemTrayIcon.DoubleClick:
        self.showNormal() # Show the interface
```

Code 3.32: Minimize to Background

The "*background_clicked()*" function creates a message box and hides the Cmail interface, adding it to the log.

The "*on_tray_icon_clicked*" function works when double-clicking on the icon in the tray menu and makes the Cmail interface visible.

```
def exit_app(self):
    msg = QMessageBox() # Create an instance of QMessageBox
    msg.setIcon(QMessageBox.Question)
    msg.setText("Are you sure you want to quit Cmail?")
    msg.setWindowTitle("Confirmation Message")
    msg.setStandardButtons(QMessageBox.Ok | QMessageBox.Cancel)
    retval = msg.exec_() # The value returned by exec_()
    if retval == QMessageBox.Ok: # If the "OK" button is pressed
        app.exit() # Exit the application
```

Code 3.33: Exit

The `exit_app()` function creates a message box with the text *"Are you sure you want to quit Cmail?"* and the title *"Confirmation Message"*. If the user clicks on the *"OK"* button, the Cmail application is exited.

Log Records

The function, defined in Code 3.34, is used to create log records. It gets a text as a parameter and adds it into the `textBrowser_logHistory` on the Logs page.

```
def add_log(self, text):
    self.uiDesign.textBrowser_logHistory.append("- " + text)
```

Code 3.34: Creating Log Record

Message Boxes

The following functions are used to create message boxes.

```
def message_box_information(self, title, text):
    msg = QMessageBox() # Create an instance of QMessageBox
    msg.setIcon(QMessageBox.Information)
    msg.setWindowTitle(title)
    msg.setText(text)
    msg.setStandardButtons(QMessageBox.Ok)
    msg.exec_() # Show the message box
```

Code 3.35: Information Message Box

The function defined in Code 3.35 is used to create message boxes for information.

```
def message_box_question(self, text, function, button, log):
    msg = QMessageBox() # Create an instance of QMessageBox
    msg.setIcon(QMessageBox.Question)
    msg.setText(text)
    msg.setWindowTitle("Confirmation Message")
    msg.setStandardButtons(QMessageBox.Yes | QMessageBox.No)
    retval = msg.exec_() # The value returned by exec_()
    if retval == QMessageBox.Yes: # If the "Yes" button is pressed
        function() # Call the function
        button(False) # Set the button
        if self.uiDesign.checkBox_notifications.isChecked():
            msg = QMessageBox() # Create an instance of QMessageBox
            msg.setIcon(QMessageBox.Information)
            msg.setText("Deleted successfully")
            msg.setWindowTitle("Confirmed Info")
            msg.setStandardButtons(QMessageBox.Ok)
            msg.exec_() # Show the message box
        self.add_log(log) # Add log record
    else:
        if self.uiDesign.checkBox_notifications.isChecked():
            msg = QMessageBox() # Create an instance of QMessageBox
            msg.setIcon(QMessageBox.Information)
            msg.setText("Cancelled by user")
            msg.setWindowTitle("Confirmed Info")
            msg.setStandardButtons(QMessageBox.Ok)
            msg.exec_() # Show the message box
```

Code 3.36: Question Message Box

This function, defined in Code 3.36, is used to create message boxes for questions. When user clicks on "Yes", the function taken as a parameter is called, and the button taken as a parameter is set to False.

If the checkbox "*checkBox_notifications*" is checked, a message box is created with the text "*Deleted successfully*" and the title "*Confirmed Info*". If user click on "No" and the checkbox "*checkBox_notifications*" is checked, another message box is created with the text "*Cancelled by user*" and the title "*Confirmed Info*".

3.4.4 LoopThread Class

This section defines "*LoopThread*", a subclass of "*QThread*" using the "*PyQt*" library. "*QThread*" is a class for creating multiple threads in PyQt applications.

Constructor of the LoopThread Class

```
def __init__(self):
    QThread.__init__(self)
    self.running = False
```

Code 3.37: Constructor of LoopThread

The "*__init__*" function, defined in Code 3.37, is the constructor function of the class. It initializes LoopThread by calling the constructor function of the QThread class. It also initializes the "*running*" property to False.

run() Function

The "*run()*" function, defined in Code 3.38, controls different events while running inside the loop. The loop runs as long as the "*running*" property is True.

```
def run(self):
    self.running = True # The thread is running
    while self.running: # While the thread is running
        try:
            cmail.start() # Call the start() function from cmail.py
        except imaplib.IMAP4.error: # If a login error occurs
            self.running = False # Terminate the loop
            self.error_signal.emit()
        if cmail.logs["email"]: # New e-mail log record
            self.email_signal.emit()
        if cmail.logs["info"]: # New information e-mail log record
            self.info_signal.emit()
        if cmail.logs["cmail"]: # New Confirmed Mail log record
            self.cmail_signal.emit()
```

Code 3.38: run()

If `"imaplib.IMAP4.error"` is caught by the try-except block in Code 3.38, the loop is terminated and `"error_signal"` is emitted. If any of the values of the `"logs["email"]"`, `"logs["info"]"` or `"logs["cmail"]"` keys are True, a signal is emitted.

stop() Function

```
def stop(self):  
    self.running = False # Terminate the loop
```

Code 3.39: stop()

When the `"stop()"` function defined in Code 3.39 is called, the `"running"` property is set to False to terminate the loop.

delete_info() & delete_email() Functions

```
def delete_info(self):  
    cmail.del_info() # Call the del_info function from cmail.py  
  
def delete_email(self):  
    cmail.del_email() # Call the del_email function from cmail.py
```

Code 3.40: delete_info()

`"delete_info"` function, defined in Code 3.40, calls the function `"del_info"` and `"delete_email"` function calls the `"del_email"` function in `"cmail.py"`.

3.4.5 Show the Application Window

```
app = QApplication([]) # Create an instance of the QApplication  
app.setAttribute(Qt.AA_EnableHighDpiScaling, True)  
window = Cmail() # Create an instance of Cmail  
window.show() # Display the window  
app.exec_() # Show the application
```

Code 3.41: Show the Application Window

The code, defined in Code 3.41, is used to show the application window.

Experimentation

This chapter discusses feedback from participants on the Confirmed Mail project.

4.1 Experiment Settings

We conducted an audio call with the participants over Discord in order to test the usability of the Cmail project and gather feedback from various participants. While we demonstrated the application to some of the participants by live streaming it on our computer, some participants tried out the demo version of the application on their personal computers and gave it a try. Then, we requested that they complete a Google Form that we had created in advance to collect their feedback and opinions.

4.2 Questions Asked

In the form we prepared, we asked the following questions to the participants:

On the first page, we asked participants a few questions about their personalities and e-mail usage, such as their age, gender, education level, the reason they use e-mail, how many active e-mail addresses they have, how frequently they check their inboxes, whether they scan incoming e-mails for fakes, and what they can do to determine whether a suspicious e-mail is fake.

On the second page, we asked participants a few questions to get their feedback on the project, such as including whether they thought it was necessary, whether they would use it, what they liked and didn't like about it, any issue they encountered while testing it, and any suggestions they had for how to make it better.

4.3 Participant Demographics

In this experiment with different participants between the ages of 18 and 24, there are a total of 20 participants, 10 males and 10 females, specifically selected to have equal numbers of both genders. Despite the fact that 80% of the participants are university students, there are also high school and university graduates.

According to the answers given, each participant has at least one e-mail address. How many active e-mail accounts they use in total is shown in the chart in Figure 4.1:

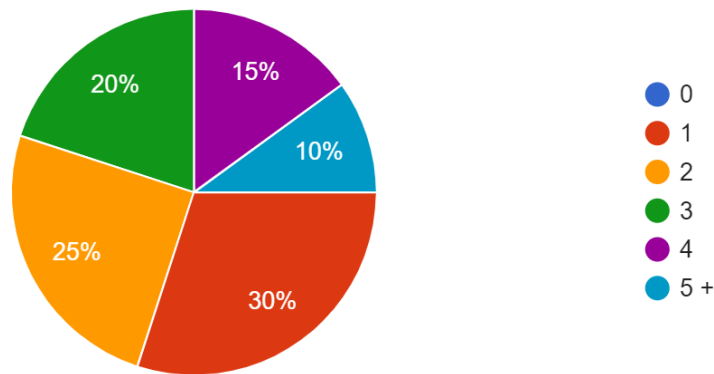


Figure 4.1: How Many E-Mail Accounts Do You Actively Use?

The majority of participants check their e-mails every day, as shown in Figure 4.2:

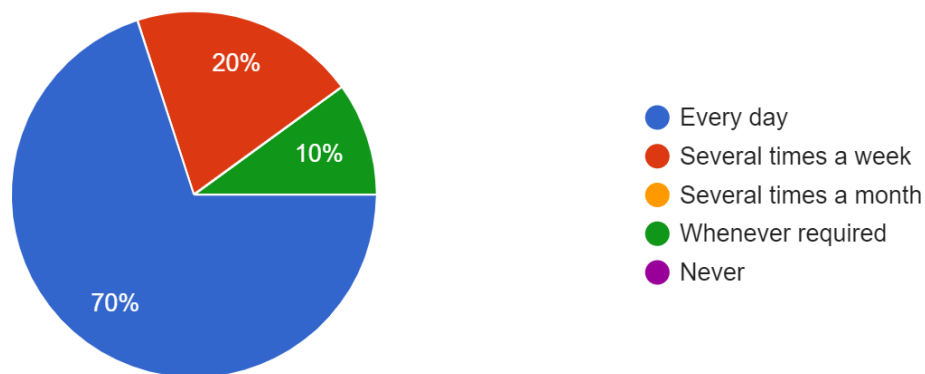


Figure 4.2: How Frequently Do You Check Your E-Mails?

The results of another question, where participants can select more than one answer, show that 85% of them use e-mail for official purposes, 70% for personal purposes, 80% to create memberships, and 50% to be informed about opportunities.

These answers demonstrate the importance of preventing e-mail based phishing attacks, as users with many accounts who frequently use e-mail for personal and official purposes are at greater risk of phishing attacks.

In spite of this risk, majority of participants do not check the legitimacy of incoming e-mails unless they are suspicious, some even do not check any incoming e-mails at all, as shown in Figure 4.3:

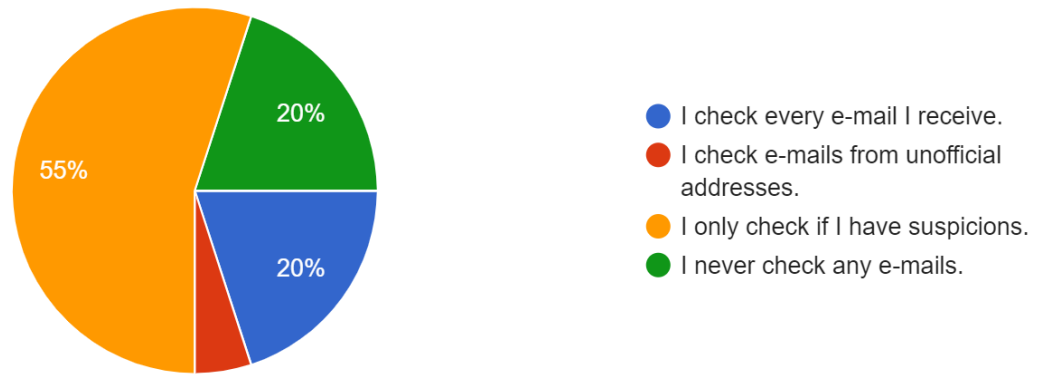


Figure 4.3: Do You Check the Legitimacy of Incoming E-Mails?

In the last question, we asked participants "If you received a suspicious e-mail, what would you do to confirm its legitimacy?". Participants are divided into three groups: those who are actively trying to take action, those acting passively because they do not know what to do, and those who are not sure what to do.

Those who do nothing would either delete the e-mail without opening it, or read it but take no action. Similarly, some of those who are undecided would do nothing, while others said they might try to look for a solution by searching on the internet.

The majority of participants who are actively trying to take action said that they would try to confirm legitimacy of incoming e-mail address. However, we have already mentioned in this thesis that e-mail addresses can be forged. Some participants said they would check the link in the e-mail. Besides these, a small percentage of participants said that they would reply to the e-mail to inform the sender and take action accordingly. This is the safest method.

As a result, a method to automatically confirm all incoming e-mails is apparently required. This requirement shows the significance of the Confirmed Mail project.

4.4 Qualitative Analysis

On the second page of the form, where we asked questions to the participants, we received feedback from them about the project. This section discusses the outcomes.

First of all, we asked whether such an application is required. 100% of participants agreed that it is required. Some of them stated that many people, including themselves, do not check the legitimacy of incoming emails. Some said that even if they check it themselves, it might be very useful for official companies. Some said that they did not need such a method because they were not aware of such a risk before. However, it is now required for all users without a doubt.

Similarly, we asked participants whether they would use such an application. 100% of the participants stated that they would use it. While some found it useful because it provided security, some said they really liked the method of use. Participants who have previously been the victim of phishing e-mails believe that using this application would not have negatively impacted them. Furthermore, some participants argue that this kind of method ought to be standard.

After that, in order to get feedback from the participants about the application, we asked them what they liked and didn't like about it. 75% of the participants stated that they did not find any weaknesses in the application. 15% thought that the user interface could be slightly improved, while 45% thought it was excellent. 10% said that the application responds to every e-mail received and this should be improved, while 50% stated that they liked the logical design of the application.

Next, we asked participants if they encountered any errors while using the application. 50% of the participants who used the application did not encounter any errors. Although most of the errors encountered by the others were fixed with their feedback, errors may still occur. These errors will be discussed in Chapter 5.

Finally, we asked the participants how the project could be improved. While 50% of the participants expressed an opinion, the other 50% found the project satisfactory and offered no suggestions. Participants who offered suggestion included, 70% suggested enhancements to the interface and increasing information provided to the user. 30% offered a few suggestions for improvements in the algorithm of the application, such as sending the confirmation code and checking incoming e-mails.

Limitations and Future Work

This chapter describes the issues encountered during the development of the Confirmed Mail project and discusses how the project might be improved in the future under various sections.

5.1 Runtime and Logic Errors

One issue that occurs when running Cmail is that the application may crash when a server communication problem occurs and may not be stable when stopped and restarted. Even though these issues are largely solved by adding delay during communication, exception handling is required for a definitive solution.

In some cases, the UTF-8 characters in the incoming e-mail content section of the response e-mail sent by Cmail might not be displayed properly. In some cases, it might also send a response to all unread e-mails instead of the most recent ones.

Cmail sends responses to no-reply e-mails, a behavior that does not follow its logic. Since these e-mails are sent automatically, they cannot reply to response e-mails sent by Cmail. Similarly, Cmail cannot see e-mails that end up in the spam box because the inbox is selected, which is another shortcoming.

5.2 Interface Enhancements

Cmail has different sizes on screens with different resolutions since the size of the interface is fixed with fixed pixels during the interface design phase.

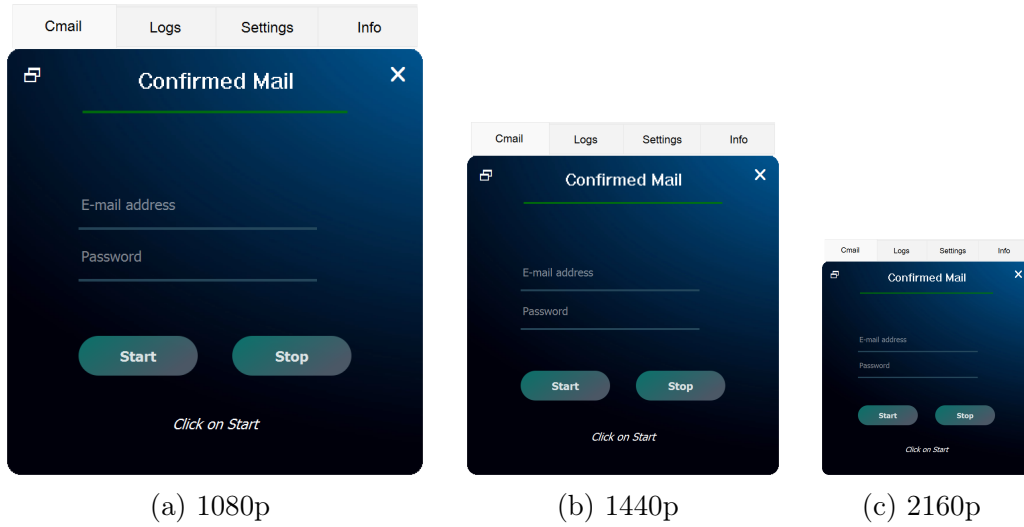


Figure 5.1: Comparison of Application Size by Resolution

As simulated in Figure 5.1, the size of the application shrinks as the number of pixels on the screen (resolution) increases. This issue might be resolved by automatically adjusting the size of the application according to the screen size or by setting different sizes for each resolution.

Apart from this, different interface settings, such as a show password button, may be added, and the position of tabs may be changed. To make the interface simpler and more contemporary for the user, it could be updated. New sections may also be added to the interface. For instance, the option to view Confirmed Mails through the interface could be provided.

5.3 Source Code Enhancements

The information text in the response e-mail sent automatically by Cmail can be made more formal and embellished with visual elements such as banners. This makes it clearer that the e-mail comes from an official application, and even a person who is not aware of this system can easily become informed of it.

Log records created while Cmail is running may be stored in a file in case the application crashes. This way, if the application crashes or if the user accidentally closes it, the log records are not lost. Moreover, in order to be able to view the Confirmed Mails mentioned in the previous section, the information about these e-mails can also be kept in a different file.

The confirmation code is generated using ready-made libraries as described in Section 3.2. The function that generates the confirmation code is in a separate file called "*authenticator.py*". The reason for this is to allow the use of a custom confirmation code generation algorithm for Cmail in the future instead of using ready-made libraries. Thus, it offers convenience for developers.

With the help of artificial intelligence, Cmail can be improved. For instance, even if incoming e-mails are confirmed, the links in them can be analyzed for maliciousness with machine learning. In addition to this, it can be used in conjunction with the methods outlined in the Chapter 1 to prevent phishing e-mails.

5.4 Integration Into E-Mail Services

The most significant advancement for future work is this concept.

Existing and potential issues could be resolved by integrating the Cmail project into e-mail services. For instance, if it is integrated in such a way that we can turn it on and off from the security settings of the e-mail provider without the need for any standalone application, the errors mentioned above about the interface can be completely fixed, and it could be presented as a faster and more reliable method.

Cmail uses the marking technique of that e-mail application to indicate confirmed e-mails. For instance, the star next to the e-mail is marked in Gmail, as shown in Figure 2.1. However, this method requires a specific marking technique for Cmail, as verified e-mails can be confused with other marked e-mails.

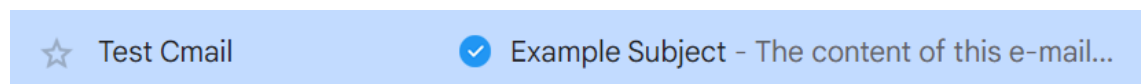


Figure 5.2: Example Confirmation Mark

A different marking method could be developed if this system was integrated with email services. In Figure 5.2, a simulation of this technique is given in an abstract form.

CHAPTER 6

Conclusion

In this thesis, we have presented the Confirmed Mail project, a robust solution designed to prevent e-mail based phishing attacks. Cmail automatically responds to incoming e-mails and sends a unique code within the response. Then, it waits for the sender to reply to the e-mail by adding this code. When the reply arrives, Cmail confirms the legitimacy of the e-mail and ensures that it comes from the same sender by checking the code contained in the reply. Thus, user-targeted phishing attacks caused by forged e-mails are successfully prevented by this system. The thesis has provided a detailed description of the Cmail application, including its user interface, usage guidelines, and comprehensive source code. Moreover, feedback from volunteer participants has been gathered, attesting to the efficacy of the application, and discussed. Additionally, issues encountered during development are addressed and potential avenues for future research and improvement are discussed. This project provides a promising solution against e-mail based phishing attacks and creates opportunities for further advancements in the field of cyber security.

References

- [1] Avi Pfeffer et al. “Artificial intelligence based malware analysis”. In: *arXiv preprint arXiv:1704.08716* (2017).
- [2] Stephen W Boyd and Angelos D Keromytis. “SQLrand: Preventing SQL injection attacks”. In: *Applied Cryptography and Network Security: Second International Conference, ACNS 2004, Yellow Mountain, China, June 8-11, 2004. Proceedings 2*. Springer. 2004, pp. 292–302.
- [3] Jibrán Saleem and Mohammad Hammoudeh. “Defense methods against social engineering attacks”. In: *Computer and network security essentials* (2018), pp. 603–618.
- [4] Peter Schaab, Kristian Beckers, and Sebastian Pape. “Social engineering defence mechanisms and counteracting training strategies”. In: *Information & Computer Security* (2017).
- [5] Ibrahim Ghafir et al. “Social engineering attack strategies and defence approaches”. In: *2016 IEEE 4th international conference on future internet of things and cloud (FiCloud)*. IEEE. 2016, pp. 145–149.
- [6] *Social Engineering*. <https://www.imperva.com/learn/application-security/social-engineering-attack/>. Accessed: 2023-05-11.
- [7] Sushruth Venkatesha, K Rahul Reddy, and BR Chandavarkar. “Social engineering attacks during the COVID-19 pandemic”. In: *SN computer science* 2 (2021), pp. 1–9.
- [8] Rachna Dhamija, J Doug Tygar, and Marti Hearst. “Why phishing works”. In: *Proceedings of the SIGCHI conference on Human Factors in computing systems*. 2006, pp. 581–590.
- [9] K Jansson and Rossouw von Solms. “Phishing for phishing awareness”. In: *Behaviour & information technology* 32.6 (2013), pp. 584–593.
- [10] Steven Furnell and Paul Dowland. *E-mail Security*. It Governance Pub, 2010.

- [11] Christine E Drake, Jonathan J Oliver, and Eugene J Koontz. “Anatomy of a Phishing Email.” In: *CEAS*. 2004.
- [12] Jordan Crain, Lukasz Opyrchal, and Atul Prakash. “Fighting phishing with trusted email”. In: *2010 International Conference on Availability, Reliability and Security*. IEEE. 2010, pp. 462–467.
- [13] Ying Zhou et al. “The effect of automation trust tendency, system reliability and feedback on users’ phishing detection”. In: *Applied Ergonomics* 102 (2022), p. 103754. ISSN: 0003-6870. DOI: <https://doi.org/10.1016/j.apergo.2022.103754>. URL: <https://www.sciencedirect.com/science/article/pii/S0003687022000771>.
- [14] Ammar Almomani et al. “A survey of phishing email filtering techniques”. In: *IEEE communications surveys & tutorials* 15.4 (2013), pp. 2070–2090.
- [15] Marcelo Luiz Brocardo et al. “Authorship verification for short messages using stylometry”. In: *2013 International Conference on Computer, Information and Telecommunication Systems (CITS)*. IEEE. 2013, pp. 1–6.
- [16] Saeed Abu-Nimeh et al. “A Comparison of Machine Learning Techniques for Phishing Detection”. In: *Proceedings of the Anti-Phishing Working Groups 2nd Annual ECrime Researchers Summit*. eCrime ’07. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2007, 60–69. ISBN: 9781595939395. DOI: 10.1145/1299015.1299021. URL: <https://doi.org/10.1145/1299015.1299021>.
- [17] Areej Alhogail and Afrah Alsabih. “Applying machine learning and natural language processing to detect phishing email”. In: *Computers Security* 110 (2021), p. 102414. ISSN: 0167-4048. DOI: <https://doi.org/10.1016/j.cose.2021.102414>. URL: <https://www.sciencedirect.com/science/article/pii/S0167404821002388>.
- [18] Xudong Pan et al. “Hidden trigger backdoor attack on {NLP} models via linguistic style manipulation”. In: *31st USENIX Security Symposium (USENIX Security 22)*. 2022, pp. 3611–3628.
- [19] *What is TCP 3 Way Handshake?* <https://www.scaler.com/topics/computer-network/tcp-3-way-handshake/>. Accessed: 2023-05-11.
- [20] Wang Aihua. “Design and optimization of two-step verification scheme for form data”. In: *Journal of Physics: Conference Series*. Vol. 1883. 1. IOP Publishing. 2021, p. 012092.