Secure Centralized Asynchronous Communication

Reese Myers
CPSC 6240
Clemson University
Clemson, South Carolina
rsmyers@clemson.edu

ABSTRACT

In this paper, we will discuss a newly developed internal mail system that caters towards smaller companies reliant on multi-user Linux systems. The program we will discuss was written in C and is known to work with the Ubuntu 22.04 distribution. We will discuss some of the dangers of multi-use email systems and explore how our program seeks to remedy several of these shortcomings.

1. INTRODUCTION

For decades, email has served as a backbone for the operation of companies of all sizes. It allows coworkers to stay up to date on recent developments and deadlines, sales teams to connect with clients, and so much more. Although some businesses promote the use of near real-time synchronous communications through applications such as Slack, email is still the bedrock of modern business. It provides an asynchronous form of communication where the user can access their messages at their leisure. Whether it's sending reports, memos, or queries, email has proven itself to be very capable. As a result, most businesses provide their employees with a dedicated email address to be used for work purposes. They expect their employees to use this company provided address to communicate with individuals and groups both inside and outside of the organization. This presents a substantial security risk. This risks the unintentional leaking of confidential company information. All it takes for information to escape from the company is an employee accidentally addressing an email to an account foreign to the company. These mail systems also frequently expose employees to phishing attacks. An email labeled “time-sensitive” or one that threatens to lock out an employee’s account may contain a link that steals the employee’s credentials if they click it. The email might seem like it’s coming from a company source but may have as little as one character off from the legitimate address. The vast majority of company email traffic is between employees of the company, so there should be a dedicated internal system for this purpose. Our application seeks to explore this issue and attempts to provide a solution that may work for some small to medium-sized businesses.

2. BACKGROUND

Many smaller companies are becoming increasingly concerned with phishing attacks. 30% of these companies consider phishing attacks to be the biggest cyber threat. 96 % of these phishing attacks are delivered via email [1]. Small companies usually do not have powerful firewall devices to keep their systems safe from these external threats. Although many common mail services, such as gmail and outlook, can be configured to tag external email as “external”, intra-message warnings can often be hidden with knowledge of CSS and html insertion [2]. Larger organizations typically have their own dedicated internal real-time communication applications. However, smaller businesses are often unable to afford custom solutions for this purpose. Some of these companies have only one server that all employees sign into to complete their work. A system that would be most beneficial in such a situation would be one that enables communication between accounts on that system. Write is a command-line utility available for Linux systems that achieves this purpose [3]. However, it is only useful for communicating with accounts that are currently logged in. Additionally, write allows for communication between devices and between networks. However, write will not solve the issues previously discussed. Write is real-time/synchronous; if a user is not logged in at the time a message is sent, it won’t be useful to them. Additionally, write is not an internal-only system. To protect against external phishing, a system needs to be made in such a way so that only those within the same network will be able to communicate.

3. MOTIVATIONS AND OBJECTIVES

Our goal for this project was to design a lightweight command-line C application that will allow for the asynchronous delivery of messages between users of a common system. We also wanted to ensure that the application integrated well with the Linux system of users and permissions. We wanted to ensure that the system was efficient and that the storage footprint of the application was minimized. In order to create a usable system to protect from external threats, we needed to provide enough features to make the system worthwhile. We needed to create a system that allows for every user to have their own instance of the application running at the same time. We also wanted clear, easy to understand navigation between message folders. Messages also needed to be timestamped and to identify senders and recipients. Messages also had to be securely located to prevent unauthorized accounts from viewing mail that did not belong to them. Attachments were another main focus. Finally, we identified ease of setup and implementation as another key component.

4. METHODOLOGY/DESIGN

4.1 Overview

The first part of our design process was to set up mailboxes for each user on the system. This means having an outbox (with a drafts folder and a sent folder) and an inbox (with an unread folder and a read folder) for each user. In order to determine the users on the system, our setup program references the /etc/passwd file and looks for usernames and UIDs representing actual (not system) users. After all the users are populated into the /CompanyMail/Config/users file, the mailbox folders for each user are created.

When a message is sent to another user, the file (and attachment if there is one) is timestamped and moved to the sent folder of the sender. Hard links of the message (and possible attachment) are then made to each destination account’s unread folder.

Logging is used in the sent, unread, and read folders to keep track of the messages that are currently in each of these specific folders. When a message is sent from one user to another, the sent log of the sender is updated with a new entry, and the unread log of each destination account is updated with a new entry as well.

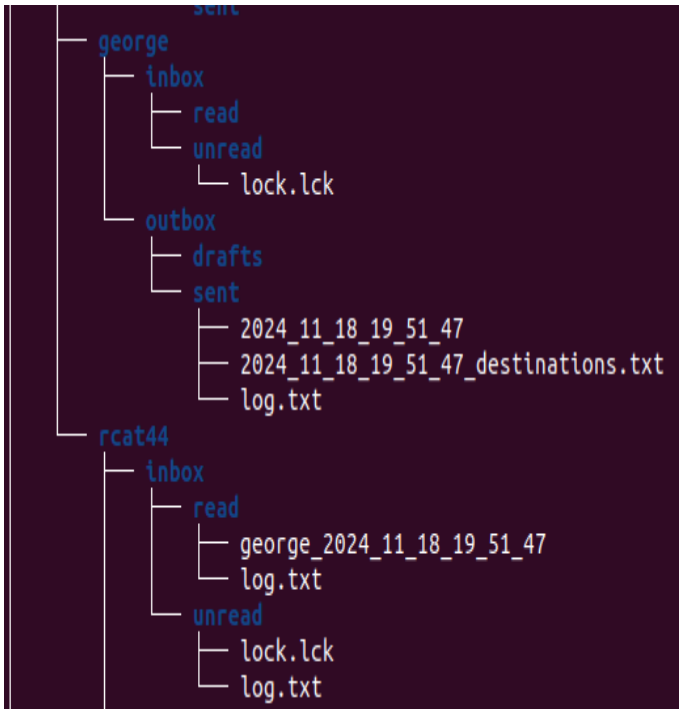

Figure 1. Example of Message Sent and Read

In Figure 1, we can see an example of a message that was sent by george to rcat44. The message, itself, is named with a timestamp, and the destination list, which lists each account the message should be sent to, is also created and placed in the sent folder. The name of the message file in the sent folder is placed into the sent log. A hard link is then made between the message in the sent folder to the destination user's (rcat44's) unread folder with the name of the sender added to the filename. In this figure, the user already read the message, so the hard link was moved to the read folder. Locks are used in unread folders to ensure that unread logs remain valid as files are received and read.

Figure 2. Regular Prompt

Figure 2 shows the menu of options provided during the normal operation of the application. A user may view unread, read, or sent messages. They might also compose a new message or quit the application

If the selection is made to compose a message, vim is opened to allow the user to create a message file. After composing the message, the user is prompted to enter a subject line. After entering the subject line, the user is asked if they would like to attach a file to their message. If yes is selected, the user is prompted to enter a path to the desired attachment file. The prompt starts with "/home/usersName/" in order to force the user to enter the name a file located within their home directory. The program checks if the user has access to the attachment file before continuing. The sender then must set a name for the file to be sent as. The recipient, when choosing to download the attachment, downloads the file as "/home/recipientUserName/providedName".

After sending a message, the sender can view the sent message, the time it was sent, and to whom it was sent. The sender can also choose to delete a sent message from the sent folder. Likewise, after reading a message, a recipient of the message can delete the message from the read folder.

4.2 Setup

Program setup is relatively easy. First, download the /CompanyMail directory from the repository. Then, simply move the /CompanyMail directory to the root of your filesystem, and execute the following commands:

- sudo chown root /CompanyMail
- sudo chmod 700 /CompanyMail

This will make the directory root owned and only accessible by users (or programs) with root-level permissions. After these commands have been executed, execute the make command within /CompanyMail. Then execute the make command within /CompanyMail/Setup. You will have to have root-level permissions when making these two make calls. As a result of these make commands, the setup utility will be ready, and the program executable will be copied to the /home directory as a root-owned SetUID program. This will allow all users on the system to run the program with root permissions as is necessary due to the protected status of the /CompanyMail folder.

To complete the setup process you will run one of the following commands:

- sudo /CompanyMail/Setup/setup
- sudo /home/mail

If the first command is used, the setup process will automatically begin. However, if the main program (mail) is executed with root-level permissions, it will open the admin setup menu and require an additional selection.

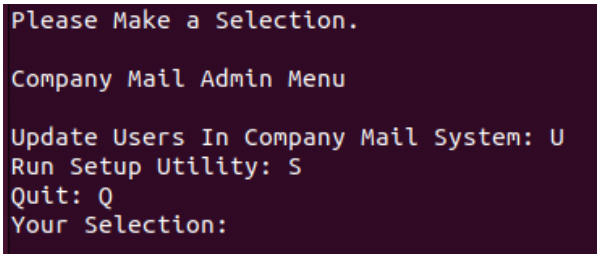

Figure 3. Admin Setup Menu

Selecting S in the admin setup menu will start the setup process and populate the users folder as previously stated. The setup program allows an administrator for the mail system to be specified. However, this administrator account does not currently have any extra privileges. The mail system administrator is welcomed as an admin when logging in. The option in the setup menu to update users allows a user running the program with root-level permissions to update the users folder to account for users that have been added to or deleted from the operating system.

5. CONCLUSION & FUTURE WORK

We were able to achieve our project's goals. Our objective was to create a centralized mail application that was isolated from external accounts. The C application that we produced was able to efficiently deliver data between individual accounts on a shared Ubuntu 22.04 Linux server.

A weakness of the program is that delivered mail is never encrypted. This was not the focus of the project because messages and attachments are merely transferred from one location to another on the system. Although, an application able to encrypt and decrypt files could be paired with this program in the future provide better security for all data kept within the /CompanyMail folder.

Another future change we would strongly consider making to improve the program is to provide more configuration options during setup, such as specifying the range of UIDs that represent users that should be added into the mail system. This would allow for deployment on a wider range of servers as they would not be limited to default OS configurations.

One last feature that we wish we had added was the ability of the administrator Company Mail account to create and alter mailing lists. With the current program, any mail addressed to a large number of recipients requires each username to be specified individually. Mailing lists would provide a massive quality-of-life improvement for users.

6. REFERENCES

[1] National University, 101 Cybersecurity Statistics and Trends for 2024. (June 2024). Retrieved November 18, 2024 from https://www.nu.edu/blog/cybersecurity-statistics/

[2] WhyNotSecurity. External Email Warning Bypass. (April 2021). Retrieved November 18, 2024 from https://whynotsecurity.com/blog/external-email-warning-bypass/

[3] IBM, Write Command. (March 2023). Retrieved November 18, 2024 from https://www.ibm.com/docs/en/aix/7.3?topic=w-write-command