Journal of Electronic Research and Application, 2024, Volume 8, Issue 4
http://ojs.bbwpublisher.com/index.php/JERA
ISSN Online: 2208-3510
ISSN Print: 2208-3502

A Design of an SMTP Email Server

Liheng Hu*

School of Cybersecurity, Beijing University of Posts and Telecommunications, Beijing 100876, China

***Corresponding author:** Liheng Hu, huliheng701@gmail.com

Abstract: This study developed a mail server program using Socket API and Python. The program uses the Hypertext Transfer Protocol (HTTP) to receive emails from browser clients and forward them to actual email service providers via the Simple Mail Transfer Protocol (SMTP). As a web server, it handles Transmission Control Protocol (TCP) connection requests from browsers, receives HTTP commands and email data, and temporarily stores the emails in a file. Simultaneously, as an SMTP client, the program establishes a TCP connection with the actual mail server, sends SMTP commands, and transmits the previously saved emails. In addition, we also analyzed security issues and the efficiency and availability of this server, providing insights into the design of SMTP mail servers.

Keywords: Mail server; Socket API; HTTP protocol; SMTP protocol; Security analysis; Efficiency analysis

Online publication: August 13, 2024

1. Introduction

With the rapid development of information technology, email has become an indispensable part of daily communication. Both businesses and individuals rely on email for quick and effective information exchange. However, existing email service providers often restrict users to specific platforms, limiting their control and customization of email services. Additionally, privacy and security concerns have increasingly become a focal point for users. In light of this, developing an independent mail server can provide greater flexibility and customization options while enhancing user control over data privacy and security [1,2].

2. Current status of email

2.1. Evolution of email services

Email services have evolved significantly from their early days of simple text transmission to the current capability of sending formatted Hyper Text Markup Language (HTML) content, including images, links, and rich text formats. This transformation has been driven by the need to support more dynamic and visually appealing communication methods. The rise of social media and instant messaging has further pushed the evolution of email services to meet user demands for more flexible and enriched forms of interaction. Modern email services now provide a platform for a wide range of multimedia content, catering to both personal and

professional communication needs [3].

2.2. Market size and trends

According to a research report, the global email market size was approximately $6.5 billion as of 2023. It is expected to grow at a compound annual growth rate (CAGR) of about 15.5% from 2024 to 2030, reaching $30.4 billion. This growth is driven by increasing levels of digitalization and the online presence of businesses and consumers, making email a cost-effective and versatile communication tool. Additionally, the number of global email users is continuously increasing, projected to reach around 4.37 billion by 2023 [4]. The sustained growth in the email market cements its importance as a versatile and effective tool in the digital marketing arsenal.

2.3. Current state of email systems

Email systems are a crucial application for transmitting communication data over the internet. Despite the advent of new communication tools, email continues to hold a significant position in both personal and business communication. It offers a low-cost, fast, and widely accepted means of communication. Furthermore, email systems support various forms of information, including text, images, and audio. This versatility makes email an indispensable communication tool, capable of handling diverse types of content and catering to a broad audience.

2.4. Security and privacy concerns

As cyber-attacks become increasingly sophisticated, the security and privacy of email services have become major concerns for both users and service providers. To protect user data and communication content, email service providers are implementing various measures, such as encryption technologies, anti-spam, and anti-phishing strategies [5,6]. These efforts aim to safeguard the integrity and confidentiality of email communications, ensuring that sensitive information remains protected from unauthorized access and malicious activities.

3. Design outline

This program is a simple proxy server designed to receive HTTP POST requests from clients. Its design approach involves creating an HTTP-based server, listening on a specific port, and extracting the email content and data from the POST requests. Subsequently, parsed email content and data are utilized to send emails to the target mailbox using the SMTP protocol. The operational workflow (Figure 1) includes receiving client requests, parsing email data, sending emails via SMTP, and error handling during the sending process. This design aims to mimic the HTTP interface of real Webmail servers, enabling communication between clients and mail servers to facilitate the sending of electronic mail [1,7].

4. Project structure

The project is mainly divided into three parts: the startup and connection of the proxy server, extraction and parsing of email content, and email sending and forwarding. The source code is divided into several modules: HTTP request handling, email sending, logging, configuration management, etc. Through Python's modular design, the code structure is clear, with distinct module functionalities, facilitating maintenance and expansion. Through the design, a simple email forwarding proxy server is implemented, capable of receiving email content from HTTP requests and forwarding it to specified email addresses. The code structure of the entire project is clear, with well-defined functional modules, making the code easy to understand and maintain.

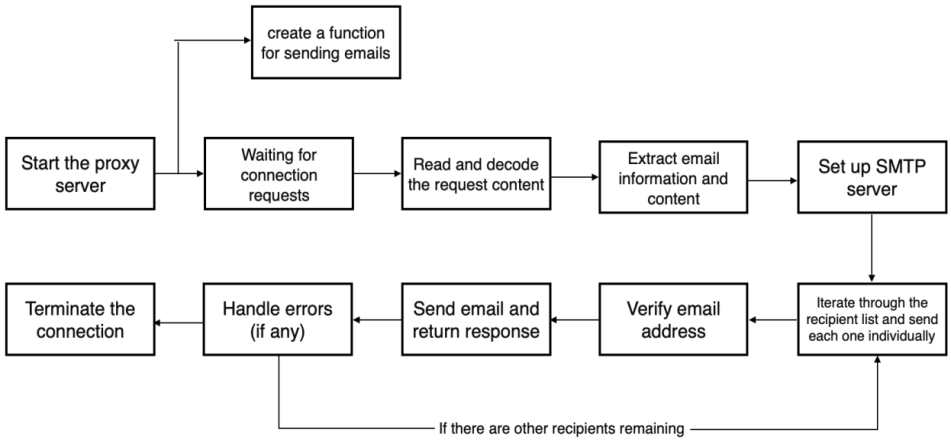

Figure 1. Email sending process

4.1. The startup and connection of the proxy server

The proxy server is created using Python's HTTPServer and BaseHTTPRequestHandler classes, listening on a specified port. When an HTTP request arrives, it parses the request content and extracts the required information, such as email content, sender, and recipient. After receiving the request, the proxy server forwards the requested content to the specified email processing module.

4.2. Extraction and parsing of email content

The server parses the email content using regular expressions or other methods, extracting recipient, sender, email subject, email content, etc. It also parses HTML-formatted email content to extract plain text content for further processing or logging.

4.3. Email sending and forwarding

Establishes a connection with the SMTP server via the SMTP protocol and logs in with authentication. Constructs the email object, including setting sender, recipient, subject, content, etc. Sends the constructed email object to the SMTP server for email forwarding. Handles possible exceptions during the email sending process, such as the recipient does not exist, the email exceeds the specified length, etc., and performs corresponding error handling.

4.4. Function test

Overall, this email server acts as a middleware between client applications generating email requests and the SMTP server responsible for delivering those emails. It handles the process of receiving, parsing, and forwarding email data, ensuring that emails are sent securely and reliably while providing error feedback to clients when necessary.

4.4.1. Email extraction and parsing

The email server functions as a proxy server, intercepting HTTP requests containing email data and forwarding them to the appropriate email processing module. Upon receiving an HTTP request, the server extracts relevant email data such as sender, recipient, subject, and email content. The server parses the email content, extracting

plain text from HTML-formatted emails and preparing it for further processing.

4.4.2. Sending and forwarding of email

The server establishes a secure connection with an SMTP server using the SMTP protocol. It then logs in to the SMTP server using the provided credentials, which include the sender's email address and password. Once authenticated, the server constructs email objects that include the sender, recipient, subject, and content information extracted from the HTTP requests. These constructed email objects are then forwarded to the SMTP server for sending to the specified recipients (Figure 2). Throughout the email sending process, the server handles potential errors (Figure 3) such as invalid recipient addresses or emails exceeding the specified length. In such cases, error messages are generated and possibly sent back to the client.

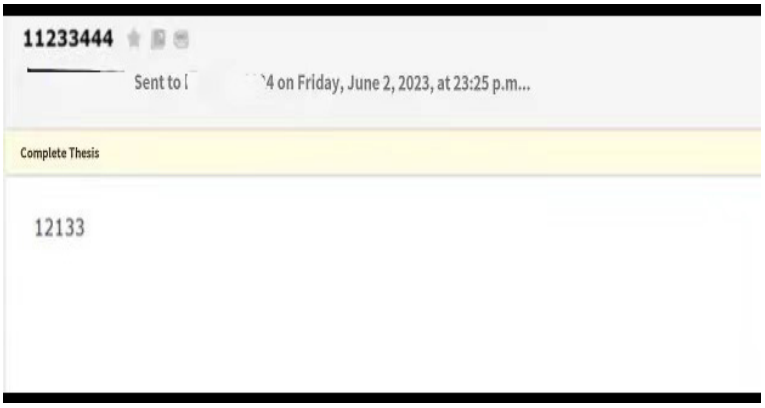

Figure 2. Email forwarded successfully

Figure 3. Incorrect email address error message

5. Analysis and improvement

5.1. Security analysis

In the design of an email server, authentication and encryption are crucial components. Ensuring that the SMTP server uses Transport Layer Security (TLS) or Secure Sockets Layer (SSL) protocols for encrypted communication effectively prevents eavesdropping and man-in-the-middle attacks, safeguarding all transmitted data, including user credentials and email contents. Similarly, employing Hypertext Transfer Protocol Secure (HTTPS) to encrypt HTTP requests further protects user data and session information. Moreover, robust

authentication mechanisms are indispensable, with strong password policies requiring users to use complex passwords and regularly change them, thereby enhancing system security. Two-Factor Authentication (2FA) adds an extra layer of security, verifying identity through codes generated by Short Message Service (SMS), email, or authentication apps, further preventing unauthorized access. For access by third-party applications, the Open Authorization (OAuth) protocol provides a secure authorization mechanism, mitigating the risk of sharing user passwords [5,6,8].

Stringent validation of all incoming data is critical to preventing injection attacks. Using regular expressions and other validation techniques to ensure incoming data conforms to expected formats effectively prevents Structured Query Language (SQL) injection and other code injection attacks. Additionally, when handling HTML content, sanitizing HTML using libraries such as Bleach or DOMPurify is essential to prevent Cross-Site Scripting (XSS) attacks. This is particularly important when parsing and displaying HTML content. Similarly, virus scanning of email attachments is essential to prevent the spread of malicious files [8].

In error handling, designing error messages that do not leak sensitive information is crucial. Avoid the disclosure of internal server details, such as stack traces or database errors. Providing generic error messages like “request processing failed” effectively prevents information leakage. Logging detailed error messages in server logs for administrator review, while ensuring the security of the logs themselves to prevent unauthorized access, is an effective security practice. Furthermore, a robust exception-handling mechanism ensures that the server remains stable and operational in the face of unforeseen errors, preventing service interruptions.

5.2. Efficiency analysis

In efficiency analysis, the modular design contributes to code maintainability and performance optimization. By separating different functionalities into independent modules, modular design not only improves code maintainability but also facilitates optimization and updates of specific functionalities. Performance analysis of various modules, especially email parsing and sending modules, identifies and optimizes performance bottlenecks. Effective system resource management is also crucial to ensuring the efficient operation of the email server. Effective memory management strategies such as object pooling and memory recycling reduce memory leaks and excessive usage. Similarly, optimizing Central Processing Unit (CPU) intensive operations, such as using more efficient algorithms and data structures, enhances overall system performance.

Implementing asynchronous processing significantly improves system concurrency. Leveraging Python’s asynchronous libraries such as asyncio increases system throughput and reduces response times. Likewise, using task queues such as Celery for background task processing effectively improves frontend response speed. In terms of scalability, ensuring system performance by adding more servers to share the load is essential. Using load balancers such as HAProxy or NGINX to distribute traffic and configuring auto-scaling policies to dynamically increase or decrease server instances based on load are effective scaling methods [9].

5.3. Maintainability and testing

When further designing the email server and officially putting it into use, maintainability and testing are also crucial factors. Adhering to coding standards and best practices, maintaining clear and readable code, and automatically checking code quality using static code analysis tools such as Pylint improves code maintainability [10]. Writing detailed documentation describing system architecture, functionality modules, and interfaces facilitates development and maintenance. Similarly, conducting regular code reviews and refactoring to identify and fix potential issues, optimize code structure, and improve readability and performance are important measures to ensure code quality. Comprehensive functional testing ensures that the server behaves as

expected in various scenarios. Implementing automated testing using tools such as Pytest for unit testing and continuous integration ensures that code changes do not introduce new issues.

5.4. Compliance

In terms of compliance, ensuring that the server complies with privacy regulations such as the General Data Protection Regulation (GDPR) is crucial. Implementing data encryption, user data access controls, and data deletion request handling effectively protects user data privacy and security. In terms of user experience, providing clear and meaningful error messages that help users understand and resolve issues while avoiding the leakage of internal details enhances user experience. Monitoring server performance and optimizing response times further improves user experience.

6. Conclusion

In terms of compliance, ensuring that the server complies with privacy regulations such as the General Data Protection Regulation (GDPR) is crucial. Implementing data encryption, user data access controls, and data deletion request handling effectively protects user data privacy and security. In terms of user experience, providing clear and meaningful error messages that help users understand and resolve issues while avoiding the leakage of internal details enhances user experience. Monitoring server performance and optimizing response times further improves user experience.

Disclosure statement

The author declares no conflict of interest.

References

[1] Sureswaran R, Al Bazar H, Abouabdalla O, et al., 2009, Active E-Mail System SMTP Protocol Monitoring Algorithm. 2009 2nd IEEE International Conference on Broadband Network & Multimedia Technology, 2009: 257–260.

[2] Dürscheid C, Frehner C, Herring SC, et al., 2013, Email communication. Handbooks of Pragmatics [HOPS], 2013(9): 35–54.

[3] Wang D, Irani D, Pu C, 2013, A Study on Evolution of Email Spam Over Fifteen Years. 9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, 2013: 1–10.

[4] Email Marketing Market Research Report: Forecast (2023–2028), n.d., viewed June 24, 2024, https://www.marknteladvisors.com/research-library/email-marketing-market.html

[5] Chonka A, Xiang Y, Zhou W, et al., 2011, Cloud Security Defence to Protect Cloud Computing Against HTTP-DoS and XML-DoS Attacks. Journal of Network and Computer Applications, 34(4): 1097–1107.

[6] Foster ID, Larson J, Masich M, et al., 2015, Security by Any Other Name: On the Effectiveness of Provider Based Email Security. Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015: 450–464.

[7] Postel JB, 1982, Simple Mail Transfer Protocol. RFC 821.

[8] LaRoche P, Zincir-Heywood AN, Heywood M, 2011, Exploring the State Space of an Application Protocol: A Case Study of SMTP. 2011 IEEE Symposium on Computational Intelligence in Cyber Security (CICS), 2011: 152–159.

[9] Naik N, 2017, Choice of Effective Messaging Protocols for IoT Systems: MQTT, CoAP, AMQP and HTTP. 2017

IEEE International Systems Engineering Symposium (ISSE), 2017: 1–7.

[10] Boogerd C, Moonen L, 2008, Assessing the Value of Coding Standards: An Empirical Study. 2008 IEEE International Conference on Software Maintenance, 2008: 277–286.