A Three-Tier Model for Addressing Man-in-the-Middle Attacks

Michelle Wang and Priya Rajbhandary
CSCI339 Distributed Systems
Professor Jeannie Albrecht
December 14th, 2024

I. Introduction

A. Project Overview

For our final project, we have decided to simulate a man-in-the-middle (MITM) attack, a cyberattack, using a three-tiered system with three key components: a client, an MITM server, and a destination server. The client sends messages to the destination server through the MITM server, which transparently intercepts and relays all communication. We modeled the implementation based on the Echo Server from Project 1, but we are using Java instead of C.

Ultimately, our goal is to demonstrate three levels of client-server interaction, showcasing progressively improved security measures. In Tier 1, plain text communication is routed through the MITM server, illustrating how unencrypted data can be easily intercepted and manipulated. In Tier 2, we introduce encryption, ensuring the confidentiality of messages, but leaving the system vulnerable to identity spoofing. Finally, in Tier 3, we incorporate SSL/TLS for authenticated communication, enabling both encryption and identity verification. Through this setup, our project hopes to emphasize the importance of combining encryption with authentication for secure communication.

II. Background and Motivation

Throughout this semester, we frequently discussed "node compromise." However, most of our readings dismissed it as being "outside the scope of the paper." So we were driven by a curiosity to simulate a man-in-the-middle (MITM) attack in a controlled and ethical way to understand the risks of interception and manipulation in a distributed system. Our goal was to visualize these vulnerabilities and demonstrate how we can circumvent these risks by making our communication channels more robust.

To achieve this, we developed a tiered security model that progresses naturally from plain-text communication to encryption, and finally to authenticated communication. This progression allows us to highlight the weaknesses at each stage and the improvements offered by combining encryption with authentication. Encryption ensures confidentiality by protecting the content of messages, while authentication ensures identity verification, preventing malicious actors from impersonating trusted parties.

We believe this is applicable to demonstrate and understand vulnerabilities in public Wi-Fi and IoT networks, where systems are more susceptible to MITM attacks. By showcasing these risks and their mitigation, our project emphasizes the importance of combining encryption with authenticated communication to secure modern networked systems.

III. System Design and Implementation

A. Architectural Overview

In our system design, the client connects to the MITM server on port 8900 and logs into the TinyBank Destination Server. After authenticating the user via a username and password, the destination server provides access to three core operations: retrieving the user's bank balance (balance), depositing funds (deposit), and withdrawing funds (withdraw). These commands are sent by the client through standard input and output streams. However, instead of being sent directly to the destination server, all messages are deliberately routed through the MITM server. This design choice allows for a

controlled and ethical simulation of a man-in-the-middle attack without relying on covert interception techniques like DNS spoofing or ARP poisoning. By explicitly configuring the client to treat the MITM server as the destination, we highlight the risks of unsecured communication channels.

The MITM server listens for client connections on port 8900 and establishes a corresponding connection to the destination server on port 8800. Acting as a proxy, the MITM server intercepts and logs messages while relaying data bi-directionally. For each client connection, it spawns a dedicated handler thread (MITMHandler) that manages two relay threads: one forwards messages from the client to the destination server, while the other forwards responses back to the client. These intercepted messages are logged in a file (mitm.log) using java.util.logging, allowing for analysis of vulnerabilities in plain-text communication.

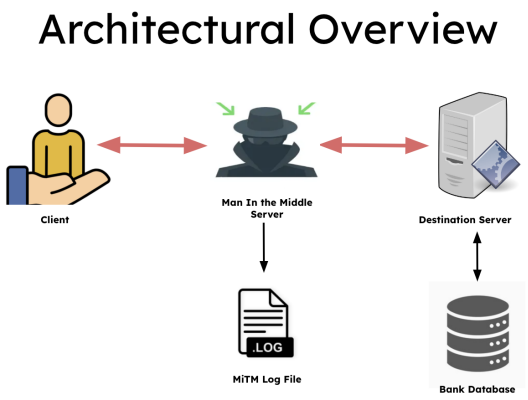

Fig. 1. System Architecture

The destination server processes requests received from the MITM server, authenticating the user against an SQLite database that securely stores client credentials, account balances, and transaction histories. It then executes the requested operations, such as retrieving account balances or updating balances for deposits and withdrawals. Synchronization is achieved using binary Semaphores to ensure thread-safe database access, especially when a user logs in from multiple clients. The destination server sends responses back to the client, initially in plain-text, to expose vulnerabilities, and later, encrypted through SSL/TLS in Tier 3. This progression demonstrates how encryption and authentication secure communication from interception and manipulation.

The system comprises three components: the client, the MITM server, and the destination server. Communication flows from the client to the destination server via the MITM server, allowing for controlled interception. The destination server authenticates users using an SQLite database and supports three operations: balance inquiries, deposits, and withdrawals.

B. Implementation Summary

This project simulates secure and insecure communication between a client, a man-in-the-middle (MITM) server, and a destination server to highlight vulnerabilities in plain-text communication and demonstrate the importance of encryption and authentication. We chose to implement the system in Java because of its intuitive syntax, robust networking, multi-threading, and cryptographic libraries, which allow for a secure and scalable design. All communication, whether plain-text or encrypted, is routed through the MITM server to demonstrate how data can be intercepted, logged, or manipulated in the absence of secure channels.

In Tier 2 and Tier 3, the client encrypts sensitive information, such as login credentials and banking commands, using AES-128 encryption implemented with the javax.crypto.Cipher class. This class enables efficient encryption and decryption, ensuring that data remains confidential during transit. In Tier 3, the system implements Authenticated Key Exchange (AKE) using SSL/TLS for secure communication between the client and the destination server. The SSL/TLS handshake establishes mutual authentication and derives a shared session key, which is used for AES encryption. The server authenticates itself by sending its certificate, which the client verifies against trusted CA certificates. Once the handshake is completed, both the client and the server use AES encryption for secure data exchange, ensuring confidentiality, integrity, and authenticity.

The MITM server acts as an intermediary, relaying intercepted messages between the client and the destination server. It logs the traffic for analysis and highlights vulnerabilities in unencrypted data exchange. The server uses java.util.logging and a FileHandler to append intercepted messages

from both directions—client to server and server to client—into a persistent log file (mitm.log). The multithreaded architecture enables the server to handle multiple client connections concurrently.

The destination server serves as the backend, managing authentication and banking operations. It securely stores user credentials and account data in an SQLite database and processes commands such as balance inquiries, deposits, and withdrawals. To ensure database security, we used prepared statements to prevent SQL injection and other backend manipulation attempts. Encryption of responses and sensitive data is achieved using the javax.crypto.Cipher class, maintaining data integrity and confidentiality. Drawing from our experience with the TinyBookStore project, we implemented binary Semaphores to synchronize banking transactions, guaranteeing thread-safe access even when a user logs in from multiple clients. The destination server employs SSL/TLS to encrypt all communication and authenticate itself to the client in Tier 3.

This project effectively highlights the risks of unsecured communication and the measures needed to mitigate them. By incorporating AES encryption via the javax.crypto.Cipher class for data confidentiality, SSL/TLS for secure data transmission, and database-level protections for integrity, the system demonstrates the evolution of security practices in networked systems. It represents the culmination of techniques learned throughout the semester, providing a comprehensive view of both vulnerabilities and solutions in secure communication.

```
Connected to server.
Enter username: alice
Enter password: password123
Server: Login successful
Enter command (or 'exit' to quit): balance
Server: Balance: 5800
Enter command (or 'exit' to quit): deposit 200
Server: Deposit successful
Enter command (or 'exit' to quit): balance
Server: Balance: 6000
Enter command (or 'exit' to quit): withdraw 7000
Server: Insufficient funds
Enter command (or 'exit' to quit): withdraw 200
Server: Withdrawal successful
Enter command (or 'exit' to quit): balance
Server: Balance: 5800
Enter command (or 'exit' to quit): exit
```

Fig. 2. Tier 1 Client-Side Interface

IV. Evaluation

We evaluated the system across three tiers of security. Each tier is described below along with its corresponding performance comparison:

- **Plain-text Communication:** Demonstrated vulnerabilities to interception and manipulation.
- **Encrypted Communication:** Secured message content using AES-128 encryption.
- **Authenticated Communication:** Combined encryption with SSL/TLS authentication to ensure confidentiality and server identity verification.

To evaluate the performance of our system across Tier 1, Tier 2, and Tier 3, we looked at four primary operations: Log In, Balance (Read Operation), Deposit (Write Operation), and Withdraw (Write Operation). The evaluation was conducted with varying levels of concurrency, specifically one, two, and three clients performing simultaneous operations. Our results highlight significant differences in response times corresponding to read and write operations on the database, number of clients, and different security mechanisms, and noticeable improvements in Tier 2 and Tier 3 over Tier 1 as concurrency increased.

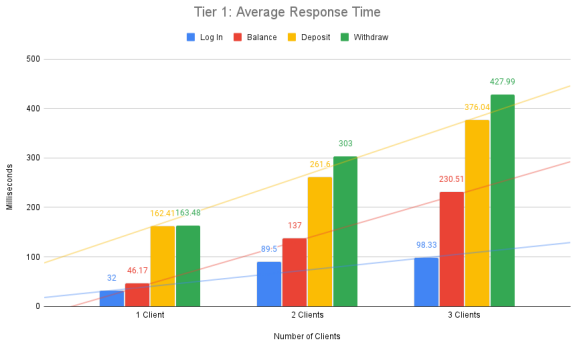

Fig. 3. Performance Comparison Tier 1

We were surprised when we discovered that Tier 1 actually showed the slowest performance, particularly for write-heavy operations like Deposit and Withdraw. We thought that because of the absence of additional encryption or authentication features, Tier 1 functions would perform faster. However, we were proven otherwise. The Deposit and Withdraw operations involve updating the database with new

account balances. For 1 client, the response times are 162.41 ms and 163.48 ms, respectively. As concurrency grows, these times increase linearly, reaching 376.04 ms for Deposit and 427.99 ms for Withdraw at 3 clients. The growing latency highlights contention and potential locking issues during database write operations due to our semaphore variable. We believe that this degradation in performance can be linked to two factors: database contention during concurrent write operations and the larger unencrypted string sizes being transferred across the network. Although the lack of encryption in Tier 1 reduces computation overhead at surface level, it actually results in higher byte size for raw strings, which can increase network latency and slow response times further.

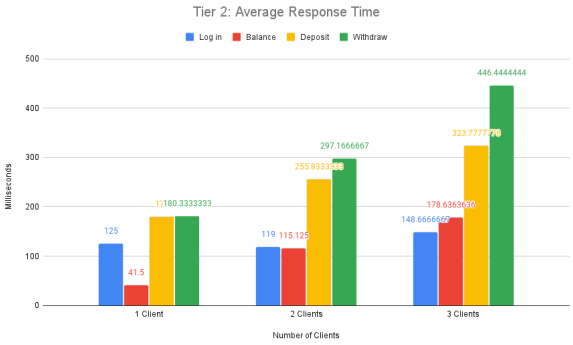

Fig. 4. Performance Comparison Tier 2

It is ironic that not only did Tier 2 improve upon data confidentiality with added encryption from Tier 1, but the Tier 2 also improved upon Tier 1's performance. It was encouraging to visualize that confidentiality and privacy do not need to compromise on performance, and seeing that encryption reduces the effective size of strings sent across the network, leading to faster data transfers. However, encryption introduces minor processing overhead, particularly noticeable during the Login function. As seen with 3 clients, the Login response times reached up to 148.67 ms. Despite this, Tier 2 achieved better scalability for both read and write operations compared to Tier 1. Balance operations reduced to 178.64 ms under 3 clients, while Deposit and Withdraw operations improved to 323.78 ms and 446.44 ms, respectively. This demonstrates that encryption, while adding computational cost, provides better overall efficiency for concurrent operations. However, within this tier, read operations still consistently outperform write operations on the database.

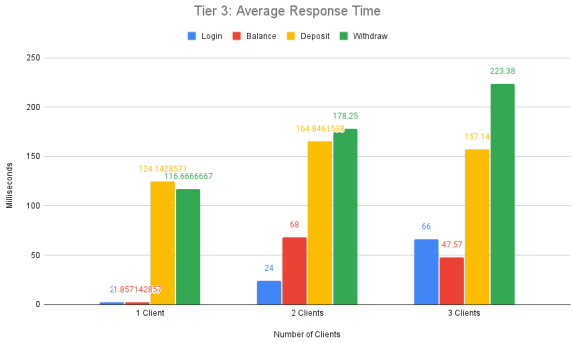

Fig. 5. Performance Comparison Tier 3

To actually test the performance of Tier 3 and visualize the performance of introducing both encryption and authentication, we removed the Man in the Middle Server, to direct our Client's destination port to 8800 i.e. the Destination Server port. We did so because only the Client and the Destination server can have an SSL/TLS authentication exchange that would verify the server's identity. We believe that the removal of the Man in the Middle Server is the reason Tier 3 achieved the best performance among the three tiers. By implementing SSL/TLS-based secure communication, Tier 3 not only ensures data encryption but also optimizes communication efficiency. This is shown in significantly lower response times across all operations, even as the number of clients increased. For example, the response time for the Login operation under 3 clients averaged just 66 ms, a significant improvement from the previous tiers. The Balance operations remained highly efficient, increasing only slightly to 47.57 ms for 3 clients, while write operations like Deposit and Withdraw averaged 157.14 ms and 223.38 ms, respectively. These results suggest that Tier 3's combination of secure communication protocols and lack of routing through the MITM Server minimizes latency and scales effectively under concurrent load.

A. Performance Summary

Overall, this evaluation shows that Tier 1 may struggle with scaling due to unoptimized string sizes and locking issues with the database during write operations. Ironically, Tier 2 mitigates some of these issues by introducing encryption, which improves overall efficiency despite a minor computational overhead. However, Tier 3 provides the best performance, combining security and scalability through SSL/TLS communication and optimized database operations. This tier demonstrates significant improvements in both read and write operations, highlighting its ability to handle concurrent client requests efficiently while maintaining data security. It was highly encouraging to discover that security does not need to come at the expense of performance and that encryption and authentication may be useful for latency optimizations. However, we discovered through the Average Response Times that write operations do not scale well across any tier, especially when concurrency is involved.

V. Future Work

While the current system effectively demonstrates vulnerabilities and security measures in communication channels, we leave room for further exploration.

One major area for further development is a front-end interface that is more intuitive for users. Instead of relying on command-line interactions, a graphical user interface (GUI) could be implemented to simulate real-world bank websites and transactions. For instance, the front-end could display intercepted messages in real-time, visualizing the risks of plaintext communication and the protection offered by encryption and authentication.

Secondly, in our current implementation, the MITM server only intercepts and logs communication between the client and the destination server. In future iterations, we could explore message modification in Tiers 1 and 2, where the MITM server can easily decrypt messages as well.

VI. Conclusions

This project demonstrates the vulnerabilities of unsecured communication channels and emphasizes the need for both encryption and authentication for any robust secure communication channel between a client and a server. Through a three-tier model, we progressively improved the system from plaintext communication to encrypted exchanges and ultimately to SSL/TLS-protected interactions. Each stage highlights its respective risks and corresponding mitigations. Our use of Java's robust libraries, such as javax.crypto.Cipher for AES encryption and javax.net.ssl for SSL/TLS, enabled the implementation of a scalable and secure system. Our integration of the SQLite database with prepared statements and semaphores further reinforced the importance of protecting sensitive data at each stage, especially in a multi-threaded system.

By deliberately simulating a controlled and ethical man-in-the-middle attack, this project emphasizes the dangers of unsecured channels and demonstrates how layered security measures – encryption, authentication, and database protections — can make a system more robust. This project is a culmination of semester-long insights, highlighting that although there may be trade-offs, a robust secure channel is the backbone of a distributed system that requires confidentiality, integrity, and reliability. Finally, it shows how security and performance complement each other in a distributed system.

References

[1] "Man-in-the-Middle Attacks." *IBM Think Blog*, 2024, https://www.ibm.com/think/topics/man-in-the-middle.

[2] "What Is an SSL/TLS Certificate?" *SSL.com*, 2024, https://www.ssl.com/article/what-is-an-ssl-tls-certificate/.