HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA AN TOÀN THÔNG TIN

BÁO CÁO BÀI THỰC HÀNH
HỌC PHẦN: THỰC TẬP CƠ SỞ
MÃ HỌC PHẦN: INT13147

BÀI THỰC HÀNH 4.1
LẬP TRÌNH CLIENT/SERVER ĐỂ TRAO ĐỔI THÔNG TIN AN TOÀN

Sinh viên thực hiện:

B22DCAT251 Đặng Đức Tài

Giảng viên hướng dẫn: TS. Phạm Hoàng Duy

HỌC KỲ 2 NĂM HỌC 2024-2025

MỤC LỤC

DANH MỤC CÁC HÌNH VẼ

CHƯƠNG 1. GIỚI THIỆU CHUNG VỀ BÀI THỰC HÀNH

1.1 Mục đích

- Giúp sinh viên hiểu về cơ chế Client/Server và có thể tự lập trình Client/Server dựa trên socket, sau đó thực hiện cài đặt giao thức đơn giản để trao đổi thông tin an toàn.

1.2 Tìm hiểu lý thuyết

1.2.1 Giới thiệu

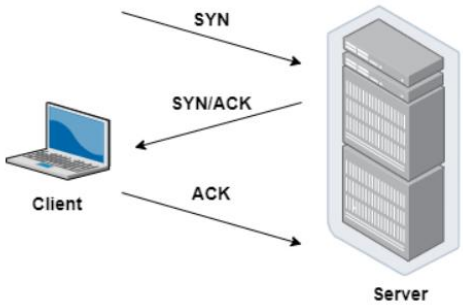
- Lập trình socket là một kỹ thuật quan trọng trong mạng máy tính, cho phép các ứng dụng giao tiếp với nhau qua mạng. Giao thức TCP (Transmission Control Protocol) là giao thức tầng giao vận phổ biến, được sử dụng trong lập trình socket nhờ tính tin cậy và khả năng đảm bảo truyền dữ liệu chính xác.

1.2.2 Tìm hiểu về Socket

- Socket là một điểm cuối (endpoint) được sử dụng để thiết lập và duy trì giao tiếp hai chiều giữa hai chương trình chạy trên mạng. Nó hoạt động như một cầu nối giữa ứng dụng và tầng giao vận trong mô hình TCP/IP, cho phép các ứng dụng gửi và nhận dữ liệu một cách hiệu quả. Trong bối cảnh TCP, socket đảm bảo một kết nối đáng tin cậy giữa client và server, giúp dữ liệu được truyền đến đúng đích, theo đúng thứ tự và không bị mất mát hay hỏng hóc. Socket có thể được hiểu như một "cổng" mà qua đó dữ liệu được truyền tải, tương tự như cách một ổ cắm điện cung cấp năng lượng cho thiết bị.
- Có nhiều loại socket phục vụ các mục đích khác nhau. Stream socket, sử dụng giao thức TCP, cung cấp một luồng dữ liệu liên tục và đáng tin cậy, phù hợp cho các ứng dụng như trình duyệt web hay email. Datagram socket, dựa trên giao thức UDP, không đảm bảo thứ tự hoặc tính toàn vẹn của dữ liệu, thường được dùng trong các ứng dụng như phát video trực tuyến, nơi tốc độ được ưu tiên hơn độ chính xác. Raw socket cho phép truy cập vào các giao thức tầng thấp hơn, nhưng ít phổ biến trong các ứng dụng thông thường do tính phức tạp và yêu cầu quyền đặc biệt.

1.2.3 Giao thức TCP

- TCP là một giao thức tầng giao vận trong mô hình TCP/IP, được thiết kế để đảm bảo truyền dữ liệu một cách chính xác và đáng tin cậy giữa hai thiết bị trên mạng. TCP hoạt động theo mô hình hướng kết nối, nghĩa là trước khi dữ liệu được gửi, một kết nối phải được thiết lập giữa client và server thông qua quy trình bắt tay ba bước (three-way handshake). Quy trình này đảm bảo rằng cả hai bên đều sẵn sàng giao tiếp và đồng bộ hóa các tham số truyền dữ liệu.

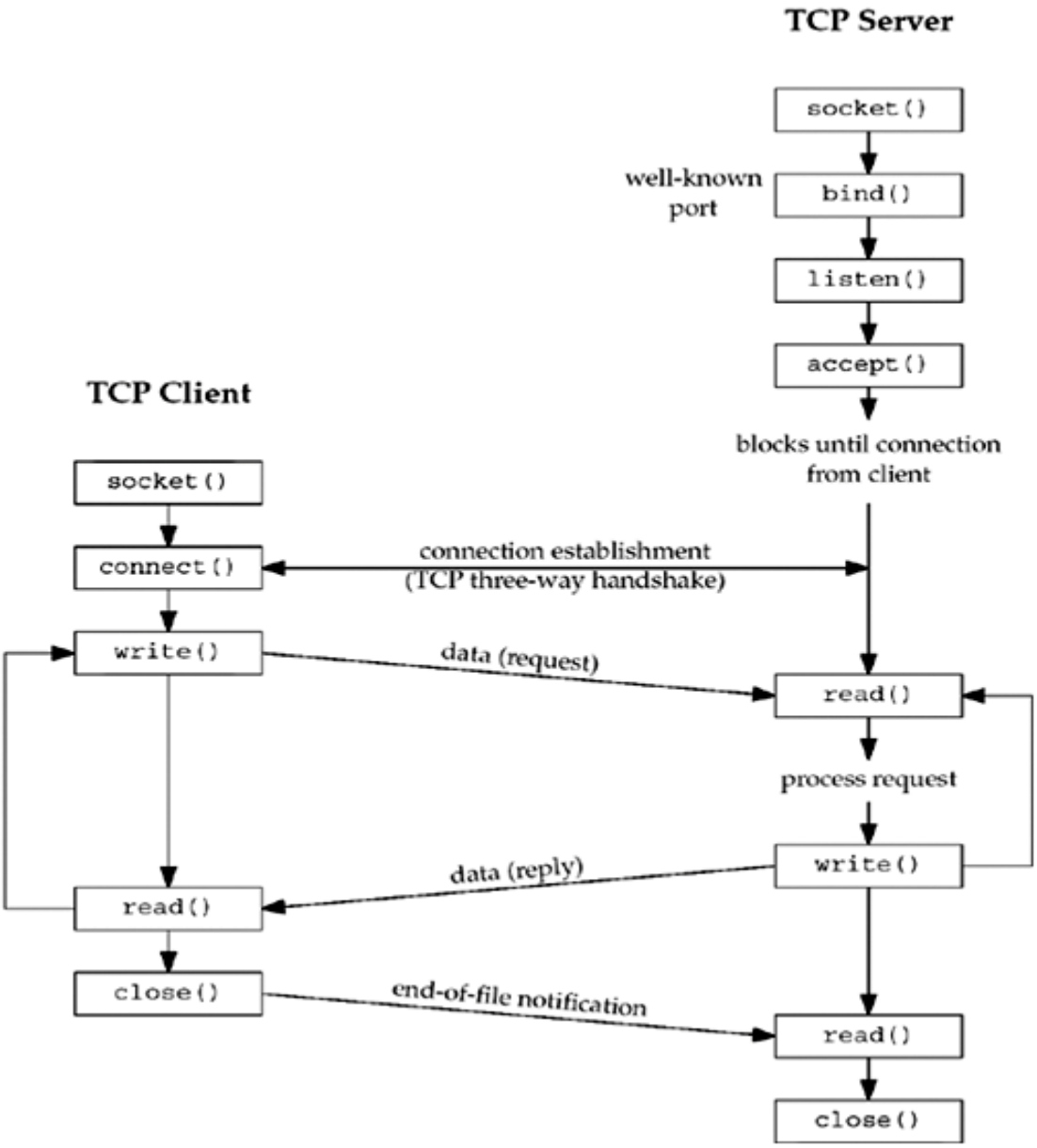


Hình 1 Quy trình bắt tay 3 bước của giao thức TCP

- TCP nổi bật với khả năng đảm bảo tính đáng tin cậy thông qua một số cơ chế. Đầu tiên, TCP sử dụng hệ thống xác nhận (acknowledgment) để kiểm tra xem dữ liệu đã được nhận chính xác hay chưa. Nếu một gói tin bị mất, TCP sẽ tự động truyền lại gói tin đó. Thứ hai, TCP kiểm soát luồng dữ liệu (flow control) để ngăn chặn việc gửi quá nhiều dữ liệu cùng lúc, tránh làm quá tải mạng hoặc thiết bị nhận. Cuối cùng, TCP sử dụng số thứ tự (sequence number) để sắp xếp các gói tin, đảm bảo dữ liệu được tái tạo theo đúng thứ tự ban đầu tại phía nhận. Những đặc điểm này làm cho TCP trở thành lựa chọn lý tưởng cho các ứng dụng yêu cầu độ chính xác cao, như truyền file hoặc giao tiếp web.

1.2.4 Quy trình hoạt động của socket TCP

- Lập trình socket với TCP tuân theo mô hình client-server, trong đó server cung cấp dịch vụ và client yêu cầu dịch vụ. Quy trình hoạt động của socket TCP được chia thành hai phần: phía server và phía client.
- Ở phía server, quá trình bắt đầu bằng việc tạo một socket để giao tiếp với các client. Socket này sau đó được gắn (bind) vào một địa chỉ IP cụ thể và một cổng (port), giúp xác định duy nhất dịch vụ trên mạng. Tiếp theo, server chuyển socket sang chế độ lắng nghe (listen), cho phép nó chờ các kết nối đến từ client. Khi một client gửi yêu cầu kết nối, server chấp nhận (accept) yêu cầu này, tạo ra một socket mới dành riêng cho việc giao tiếp với client đó. Qua socket này, server và client có thể trao đổi dữ liệu, ví dụ như gửi và nhận thông điệp. Khi quá trình giao tiếp hoàn tất, cả hai bên đóng socket để giải phóng tài nguyên.
- Ở phía client, quy trình đơn giản hơn. Client bắt đầu bằng việc tạo một socket, sau đó sử dụng địa chỉ IP và cổng của server để thiết lập kết nối (connect). Một khi kết nối được thiết lập, client có thể gửi dữ liệu đến server và nhận dữ liệu trả về. Sau khi hoàn tất giao tiếp, client đóng socket để kết thúc phiên làm việc. Quy trình này đảm bảo rằng dữ liệu được truyền qua mạng một cách an toàn và có tổ chức.



Hình 2 Quy trình TCP Socket

CHƯƠNG 2. NỘI DUNG THỰC HÀNH

2.1 Chuẩn bị môi trường

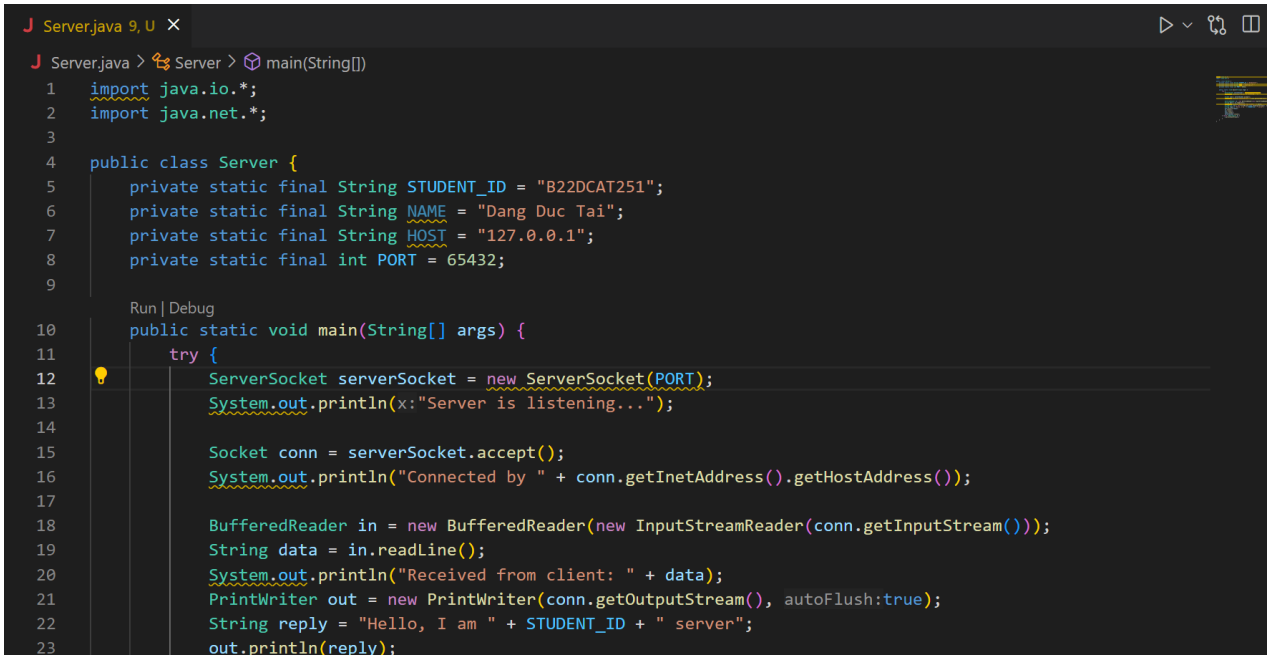
- Môi trường Python hoặc Java để chạy được ứng dụng Client/Server đã lập trình.
- Phần mềm Wireshark

2.2 Các bước thực hiện

2.2.1 Lập trình Client và Server với TCP socket

2.2.1.1 Coding

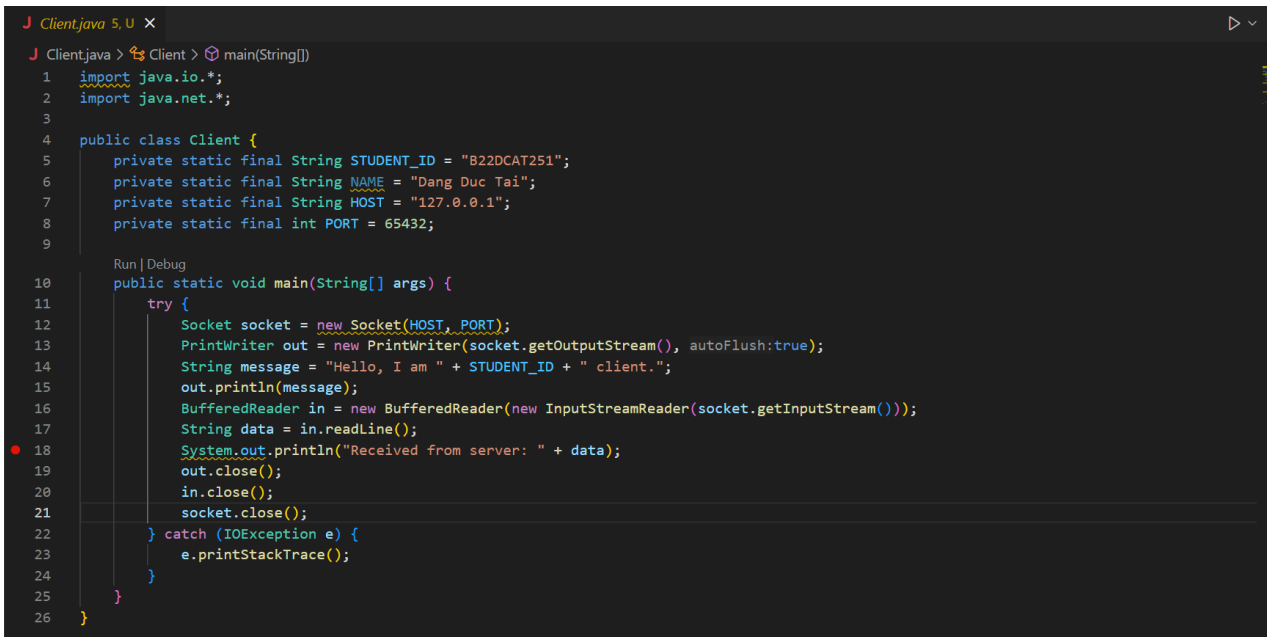
- Lập trình Server
- Sử dụng thư viện java.net, ServerSocket chạy tại cổng 65432
- Thông điệp được truyền trực tiếp giữa Server - Client



Hình 3 Code Java Server (1)

Hình 4 Code Java Server (2)

- Lập trình Client

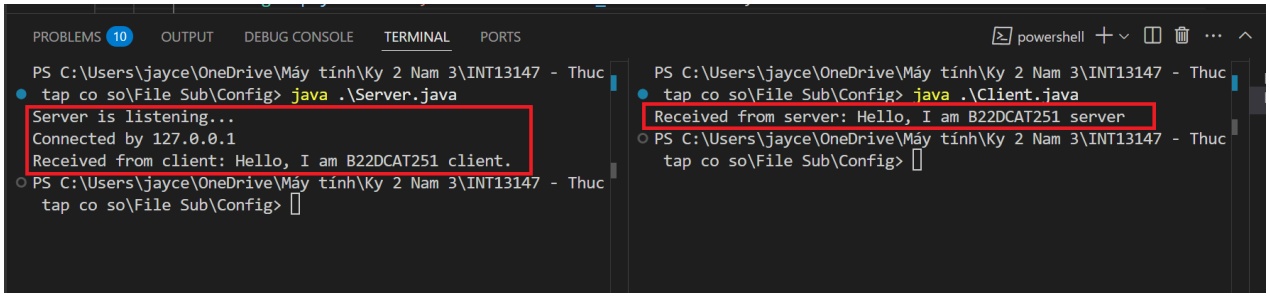


```java
import java.io.*;
import java.net.*;

public class Client {
    private static final String STUDENT_ID = "B22DCAT251";
    private static final String NAME = "Dang Duc Tai";
    private static final String HOST = "127.0.0.1";
    private static final int PORT = 65432;

    Run | Debug
    public static void main(String[] args) {
        try {
            Socket socket = new Socket(HOST, PORT);
            PrintWriter out = new PrintWriter(socket.getOutputStream(), autoFlush:true);
            String message = "Hello, I am " + STUDENT_ID + " client.";
            out.println(message);
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            String data = in.readLine();
            System.out.println("Received from server: " + data);
            out.close();
            in.close();
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Hình 5 Code Java Client

2.2.1.2 Trao đổi thông điệp

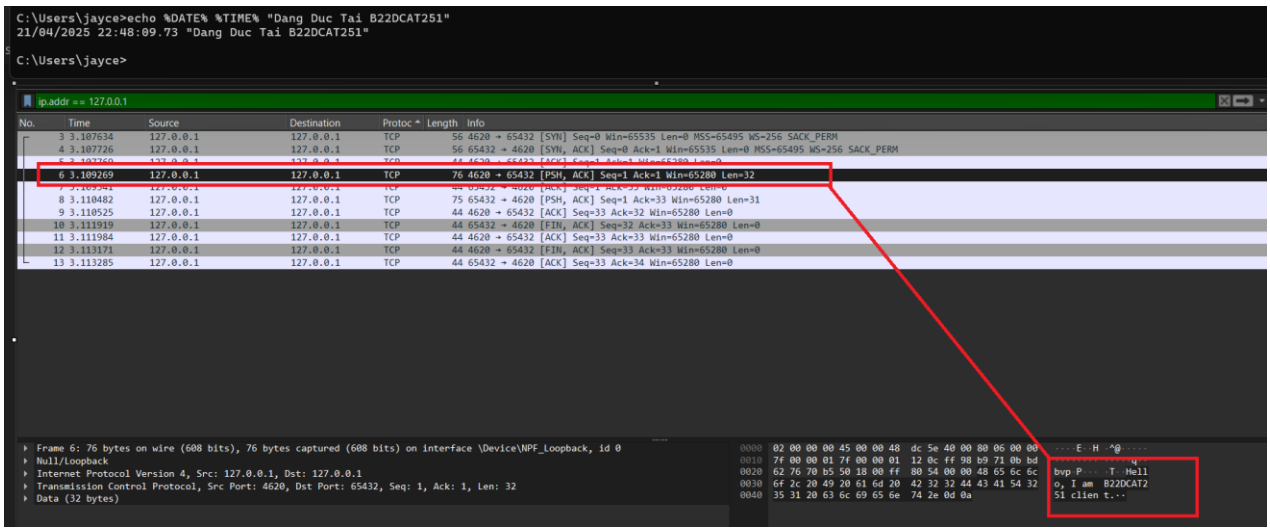
- Chạy code Server, sau đó chạy Client. Client gửi thông điệp cá nhận hóa cho Server: "Hello, I am B22DCAT251 Client."
- Server nhận được hiển thị thông điệp nhận được và gửi lại Client thông điệp: Server gửi lại "Hello, I am B22DCAT251 Server"
  java Server.java
  java Client.java



Hình 6 Chạy code Java Socket

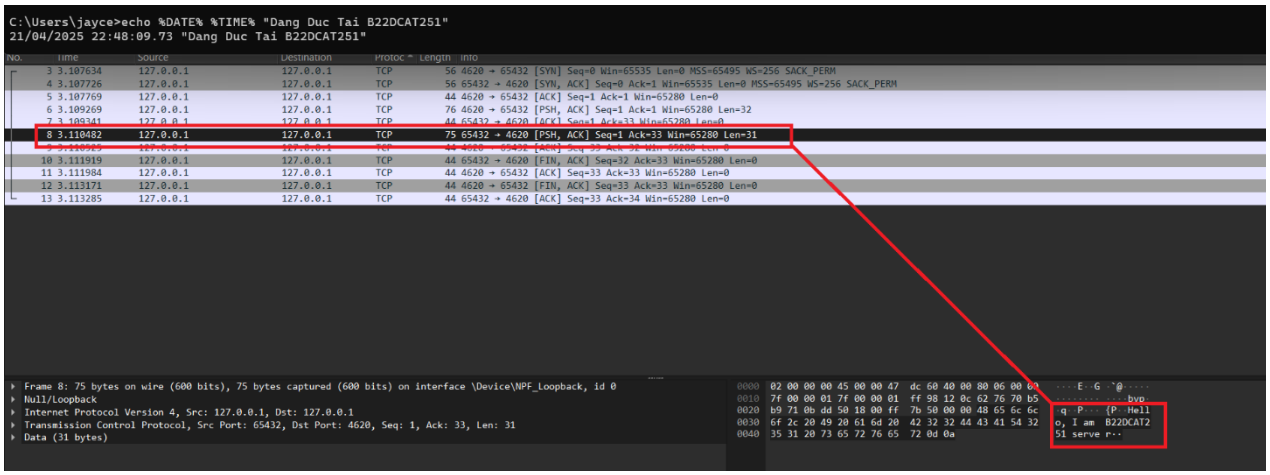
2.2.1.3 Bắt gói tin với wireshark

- Sử dụng Wireshark để bắt các thông tin đã gửi từ Client đến Server và ngược lại
- Bắt gói tin từ Client



Hình 7 Bắt gói tin từ Client trên wireshark

- Bắt gói tin từ Server

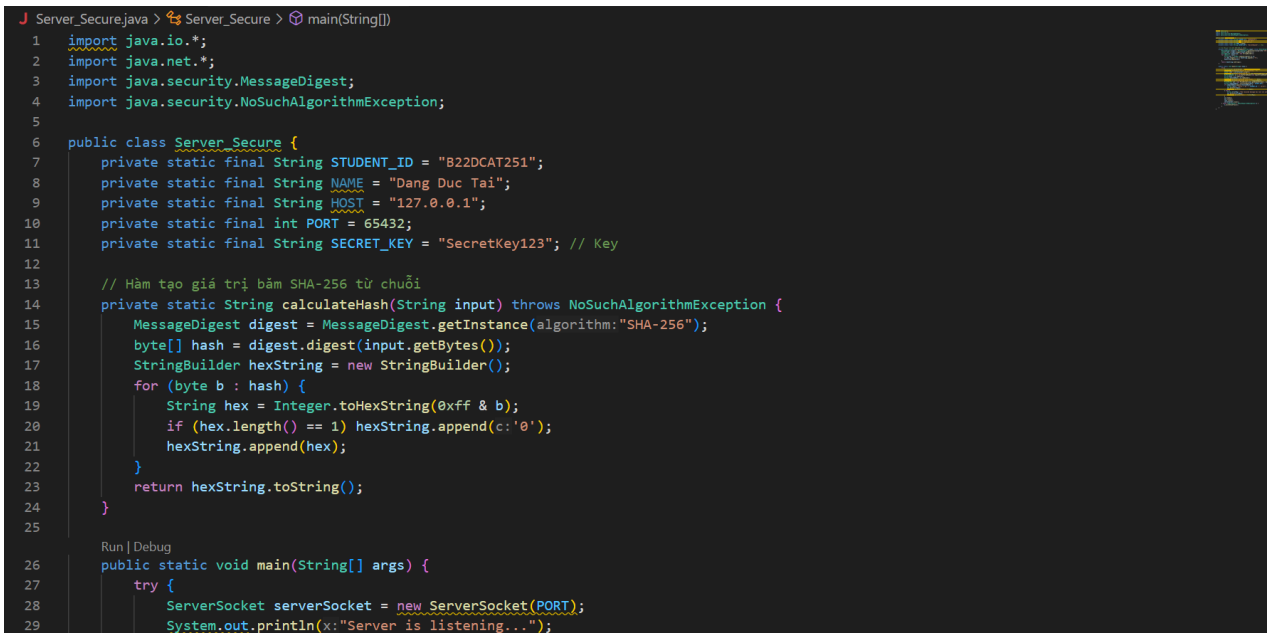


Hình 8 Bắt gói tin từ Server trên wireshark

2.2.2 Lập trình đảm bảo tính toàn vẹn

2.2.2.1 Coding

- Code Server
- Tương tự như phần trước, chỉ thay đổi bằng cách thêm thuật toán kiểm tra tính toàn vẹn SHA-256 với SECRET_KEY đã thống nhất từ trước
- Thuật toán sẽ băm SHA-256 chuỗi key trên Server, sau đó kiểm tra có khớp với hash từ Client gửi đến hay không
- Hash được tính bằng data+key



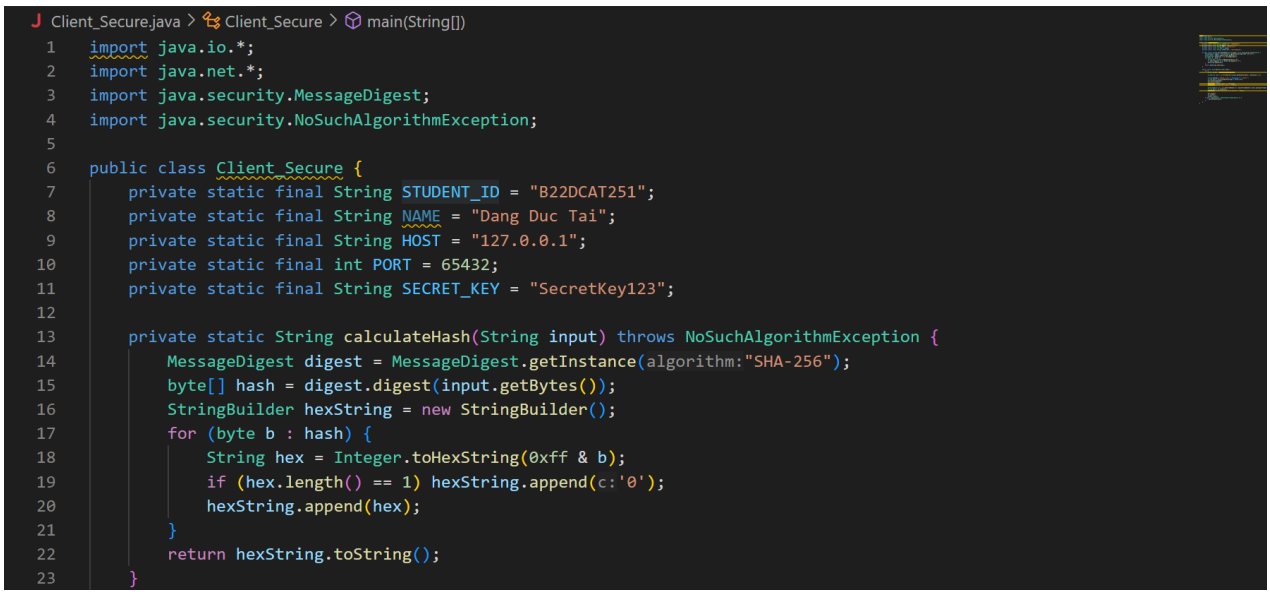
```java
import java.io.*;
import java.net.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class Server_Secure {
    private static final String STUDENT_ID = "B22DCAT251";
    private static final String NAME = "Dang Duc Tai";
    private static final String HOST = "127.0.0.1";
    private static final int PORT = 65432;
    private static final String SECRET_KEY = "SecretKey123"; // Key

    // Hàm tạo giá trị băm SHA-256 từ chuỗi
    private static String calculateHash(String input) throws NoSuchAlgorithmException {
        MessageDigest digest = MessageDigest.getInstance(algorithm:"SHA-256");
        byte[] hash = digest.digest(input.getBytes());
        StringBuilder hexString = new StringBuilder();
        for (byte b : hash) {
            String hex = Integer.toHexString(0xff & b);
            if (hex.length() == 1) hexString.append(c:'0');
            hexString.append(hex);
        }
        return hexString.toString();
    }

    Run | Debug
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(PORT);
            System.out.println(x:"Server is listening...");
```

Hình 9 Code Java Server Secure (1)

```java
public class Server_Secure {
    public static void main(String[] args) {
            System.out.println(x:"Server is listening...");
            Socket conn = serverSocket.accept();
            System.out.println("Connected by " + conn.getInetAddress().getHostAddress());
            BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
            String data = in.readLine();
            String receivedHash = in.readLine();
            System.out.println("Received from client: " + data);
            System.out.println("Received hash: " + receivedHash);
            PrintWriter out = new PrintWriter(conn.getOutputStream(), autoFlush:true);
            String computedHash = calculateHash(data + SECRET_KEY);
            if (computedHash.equals(receivedHash)) {
                String reply = "Hello, I am " + STUDENT_ID + " server";
                out.println(reply);
                System.out.println("Sent: " + reply);
            } else {
                String errorMsg = "The received message has lost its integrity.";
                out.println(errorMsg);
                System.out.println("Sent: " + errorMsg);
            }
            in.close();
            out.close();
            conn.close();
            serverSocket.close();
        } catch (IOException | NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }
```

Hình 10 Code Java Server Secure (2)

- Code Client



```java
import java.io.*;
import java.net.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class Client_Secure {
    private static final String STUDENT_ID = "B22DCAT251";
    private static final String NAME = "Dang Duc Tai";
    private static final String HOST = "127.0.0.1";
    private static final int PORT = 65432;
    private static final String SECRET_KEY = "SecretKey123";

    private static String calculateHash(String input) throws NoSuchAlgorithmException {
        MessageDigest digest = MessageDigest.getInstance(algorithm:"SHA-256");
        byte[] hash = digest.digest(input.getBytes());
        StringBuilder hexString = new StringBuilder();
        for (byte b : hash) {
            String hex = Integer.toHexString(0xff & b);
            if (hex.length() == 1) hexString.append(c:'0');
            hexString.append(hex);
        }
        return hexString.toString();
    }
```

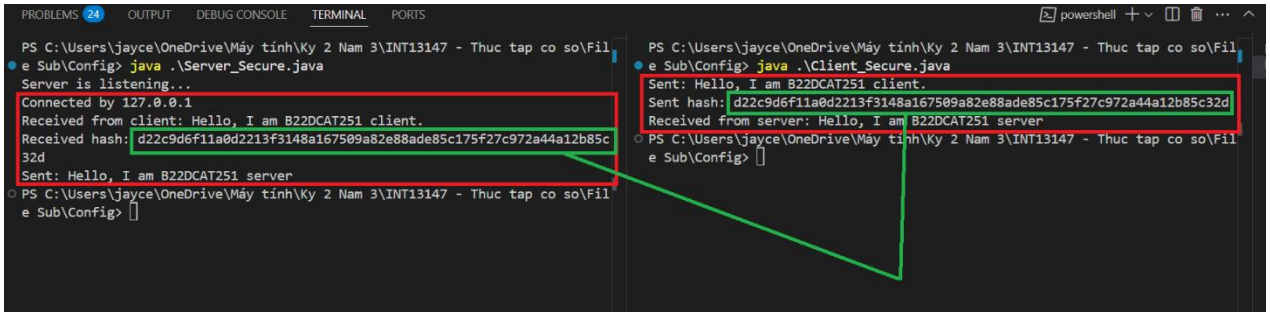
Hình 11 Code Java Client Secure (1)

```java
    public static void main(String[] args) {
        try {
            Socket socket = new Socket(HOST, PORT);

            PrintWriter out = new PrintWriter(socket.getOutputStream(), autoFlush:true);

            String message = "Hello, I am " + STUDENT_ID + " client.";
            String hash = calculateHash(message + SECRET_KEY);
            out.println(message);
            out.println(hash);
            System.out.println("Sent: " + message);
            System.out.println("Sent hash: " + hash);

            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            String data = in.readLine();
            System.out.println("Received from server: " + data);

            out.close();
            in.close();
            socket.close();
        } catch (IOException | NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }
}
```

Hình 12 Code Java Client Secure (2)

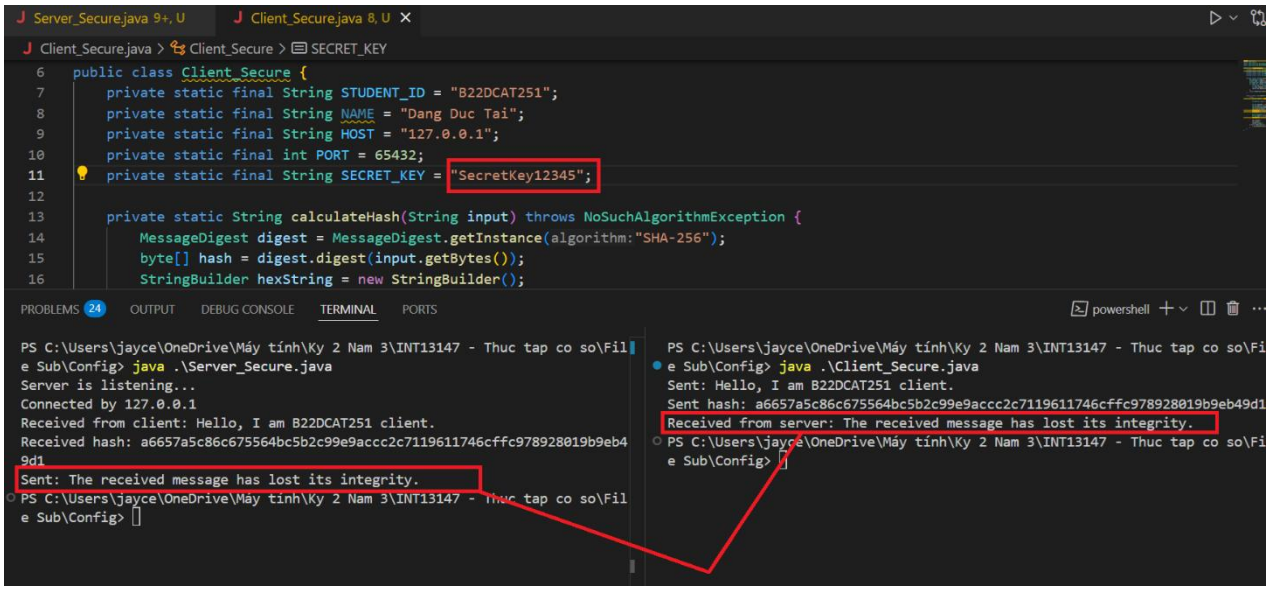
2.2.2.2 Trao đổi thông điệp

- Từ Client và Server, sửa đổi để sao cho: khi gửi thông điệp sẽ gửi kèm theo giá trị băm của (data+key) để phía bên kia kiểm tra xác minh tính toàn vẹn. Hai bên có thể thống nhất một giá trị key trước đó.



Hình 13 Trao đổi thông điệp với khóa bí mật

- Thay đổi giá trị key tại Client và thực hiện gửi lại, nếu không đáp ứng tính toàn vẹn cần thông báo: “The received message has lost its integrity.”

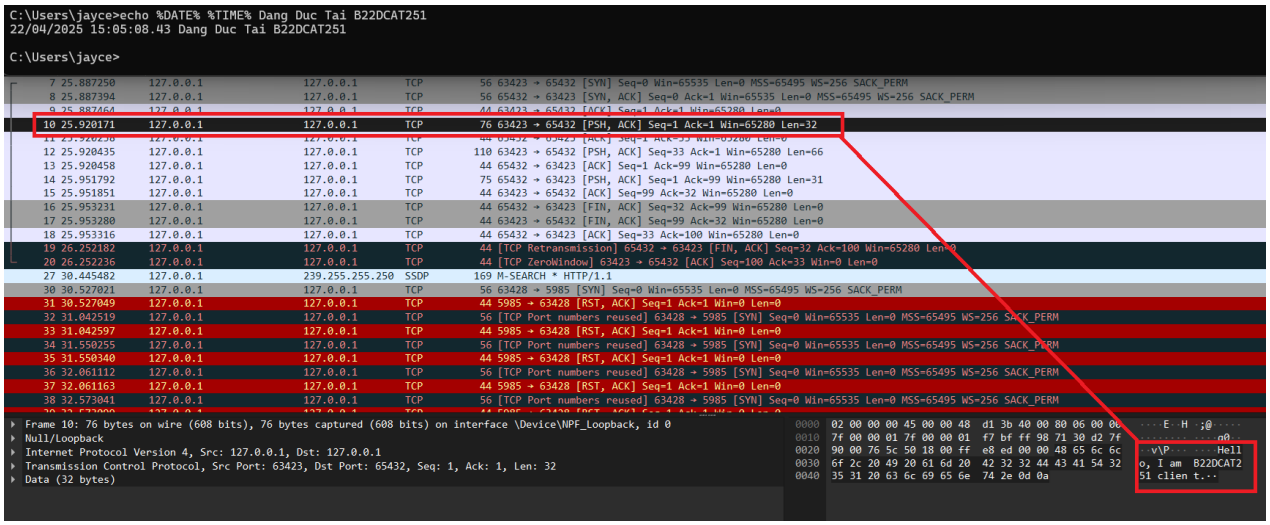


Hình 14 Thông báo lỗi khi thông tin không được xác thực

2.2.2.3 Bắt gói tin với Wireshark

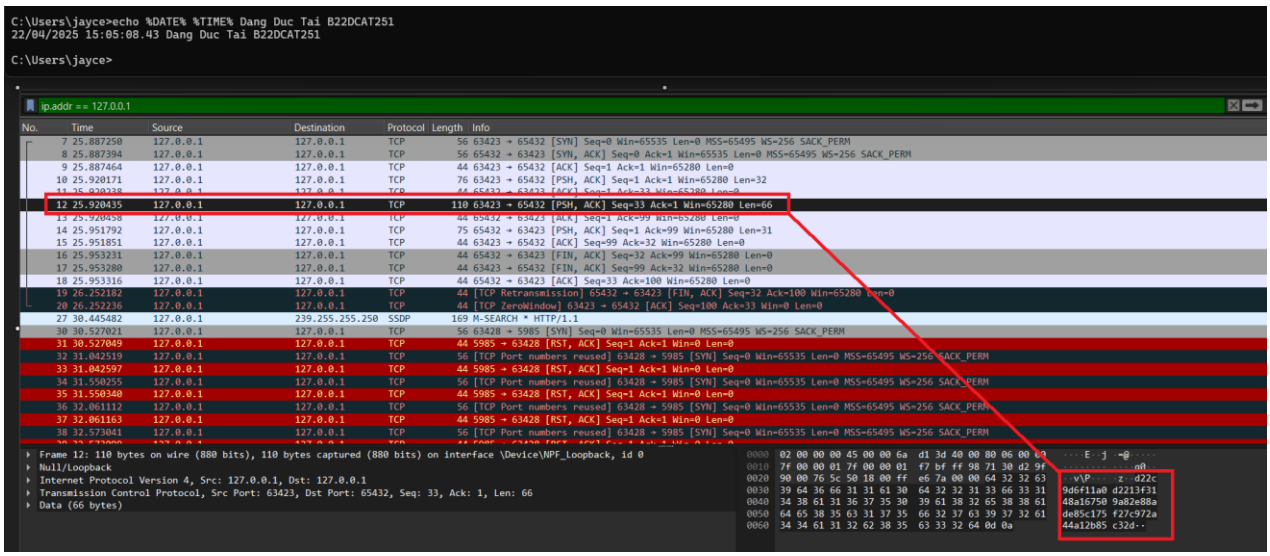
- Bắt được các bản tin trao đổi giữa Client và Server trong Wireshark

- Gói tin từ Client



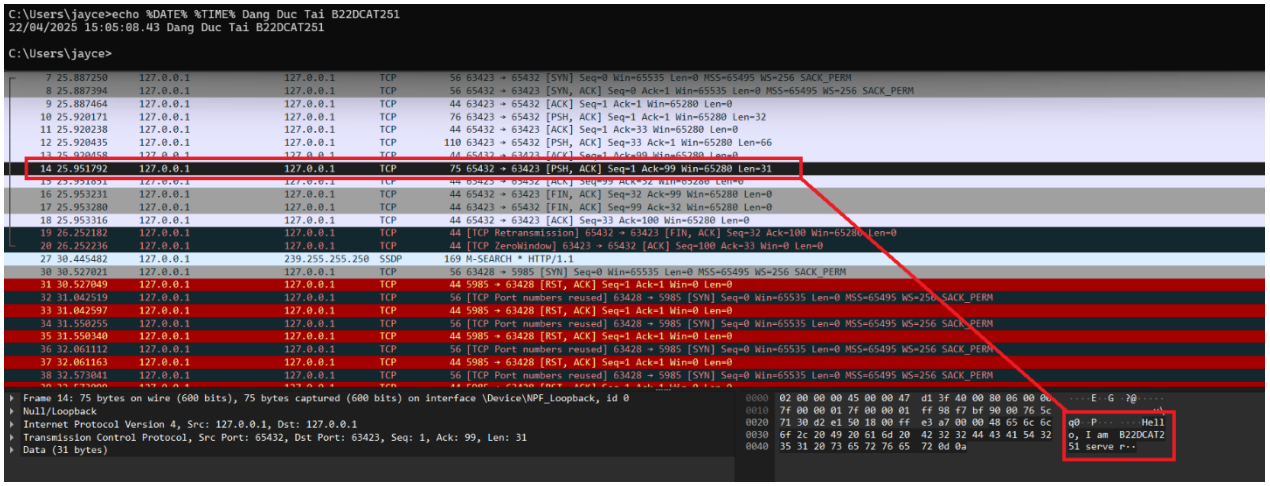
Hình 15 Bắt gói tin từ Client trên wireshark

- Gói tin yêu cầu xác thực hàm băm



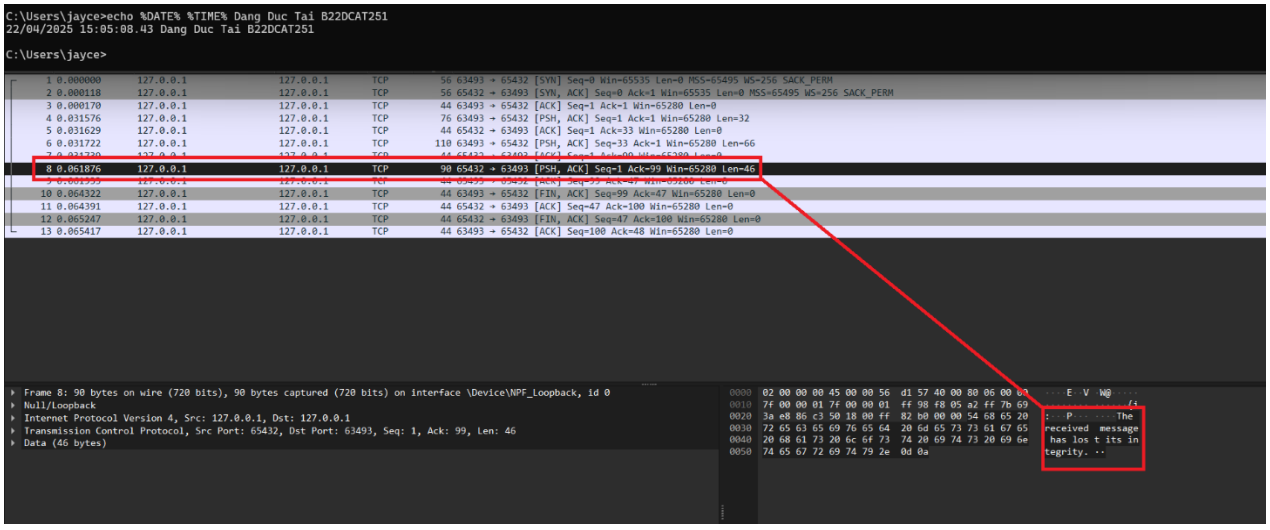
Hình 16 Bắt gói tin xác thực hàm băm

- Gói tin từ Server



Hình 17 Bắt gói tin từ Server trên wireshark

- Gói tin khi truyền bị lỗi xác thực



Hình 18 Bắt gói tin truyền bị lỗi xác thực

TÀI LIỆU THAM KHẢO

[1] Chapter 2: Application Layer V8.1 (9/2020) http://gaia.cs.umass.edu/kurose_ross/ppt.php.