Modul 6

Socket Programming Security

Tugas Pendahuuan :

1. Mengapa keamanan menjadi aspek penting dalam implementasi Socket Programming?
2. Apa itu verifikasi input dan mengapa penting dalam mencegah ancaman keamanan?
3. Jelaskan mengapa validasi server sangat penting dalam konteks Socket Programming.
4. Bagaimana validasi server dapat membantu mencegah identitas palsu dan akses yang tidak sah?
5. Sebutkan dua teknik atau langkah yang dapat diambil untuk melindungi data yang dikirimkan melalui koneksi socket.
6. Apa peran autentikasi dalam implementasi keamanan Socket Programming?

Materi :

Socket Programming Security adalah upaya untuk melindungi aplikasi yang menggunakan TCP Socket Programming dari serangan yang dapat merusak keamanan atau privasi data. Dalam pemrograman socket, ada beberapa hal yang dapat dilakukan untuk meningkatkan keamanan aplikasi:

1. Verifikasi Input: Pastikan bahwa input yang diterima dari pengguna atau jaringan adalah valid dan sesuai dengan yang diharapkan. Hindari penggunaan input langsung dalam operasi yang dapat membuka celah keamanan.
2. Validasi input server: Pastikan bahwa server hanya menerima koneksi dari host yang valid dan terpercaya. Ini dapat dilakukan dengan menggunakan teknik autentikasi atau firewalling.
3. Enkripsi: Data yang dikirim melalui jaringan harus dienkripsi untuk menghindari pencurian data atau sniffing paket oleh pihak yang tidak berwenang. Protokol SSL / TLS atau SSH dapat digunakan untuk enkripsi data.
4. Validasi server: Pastikan bahwa koneksi terjadi hanya dengan server yang valid dan terpercaya. Hal ini dapat dicapai dengan teknik autentikasi atau menggunakan sertifikat digital.
5. Membatasi hak akses: Batasi hak akses pada pengguna atau host yang tidak diizinkan untuk memperoleh akses pada aplikasi yang berjalan di server.
6. Melakukan filtering: Lakukan filtering terhadap koneksi yang diterima dan kirimkan hanya pesan yang valid ke aplikasi. Hal ini dapat membantu mencegah serangan yang ditujukan pada aplikasi.
7. Monitoring dan logging: Pantau semua aktivitas di dalam aplikasi dan lakukan logging terhadap segala aktivitas yang mencurigakan atau berpotensi merugikan keamanan aplikasi.

Dalam mengimplementasikan keamanan pada aplikasi socket di Java, dapat digunakan library seperti Bouncy Castle atau Java Security Architecture (JSA). Selain itu, dapat juga menggunakan library pihak ketiga seperti Spring Security atau Apache Shiro untuk mengamankan aplikasi socket.

Verifikasi Input

Untuk memverifikasi input yang diterima dari sisi aplikasi client, dapat dilakukan beberapa langkah sebagai berikut:

1. Validasi data: Pastikan data yang diterima dari sisi client telah memenuhi persyaratan yang diperlukan, seperti format data, tipe data, panjang data, dan sebagainya. Jika data tidak valid, dapat mengirimkan pesan kesalahan ke sisi client.
2. Sanitasi data: Pastikan data yang diterima dari sisi client aman dan tidak mengandung karakter yang berbahaya atau dapat mengeksploitasi kerentanan sistem. Dalam hal ini, dapat menggunakan teknik sanitasi data seperti escape character atau input filtering.
3. Autentikasi pengguna: Pastikan bahwa pengguna yang melakukan permintaan atau mengirim data ke server telah terotentikasi dan memiliki hak akses yang sesuai. Dalam hal ini, dapat menggunakan teknik autentikasi seperti login atau OAuth.
4. Enkripsi data: Jika data yang diterima dari sisi client sensitif, seperti informasi login atau data pribadi, sebaiknya dienkripsi untuk mencegah peretasan data saat data sedang dalam perjalanan dari client ke server.
5. Validasi input parameter: Pastikan parameter input yang diterima dari sisi client tidak mengandung karakter yang tidak valid dan dapat mengeksploitasi kerentanan sistem. Dalam hal ini, dapat menggunakan teknik validasi input parameter seperti whitelist dan blacklist.

Dalam implementasinya, langkah-langkah tersebut dapat diimplementasikan dalam kode program dengan menggunakan library atau framework yang sesuai, seperti Apache Shiro, Spring Security, atau OWASP ESAPI.

Berikut ini adalah contoh implementasi verifikasi input dalam aplikasi client menggunakan Java:

Client

```
import java.util.Scanner;

public class Client {
    public static void main(String[] args) {
        try {
            Scanner scanner = new Scanner(System.in);
            System.out.print("Masukkan input: ");
            String input = scanner.nextLine();

            // Verifikasi input
            if (input.matches("[a-zA-Z0-9]+")) {
                System.out.println("Input valid!");
                // Lakukan operasi lainnya jika input valid
            } else {
                System.out.println("Input tidak valid!");
                // Lakukan tindakan lain jika input tidak valid
            }
        } catch (Exception e) {
            System.out.println("Terjadi kesalahan: " + e.getMessage());
        }
    }
}
```

Dalam contoh di atas, input yang diterima dari aplikasi client akan diverifikasi menggunakan regular expression. Jika input sesuai dengan pola yang ditentukan dalam regular expression, maka input dianggap valid dan program akan melanjutkan operasi selanjutnya. Namun, jika input tidak sesuai dengan pola, maka program akan menampilkan pesan bahwa input tidak valid dan melakukan tindakan yang sesuai.

Berikut adalah contoh implementasi dalam Java untuk pengiriman pesan dari sisi server menggunakan socket dan menerima pesan di sisi client:

Sisi Server (pengirim):

Server

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

public class Server {
    public static void main(String[] args) throws IOException {
        int port = 12345;
        DatagramSocket socket = new DatagramSocket();
        InetAddress address = InetAddress.getByName("localhost");

        BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
        while (true) {
            String message = reader.readLine();
            byte[] buffer = message.getBytes();
            DatagramPacket packet = new DatagramPacket(buffer, buffer.length,
address, port);
            socket.send(packet);
        }
    }
}
```

Sisi Klien

Client.java

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

public class Client {
    public static void main(String[] args) throws IOException {
        int port = 12345;
        DatagramSocket socket = new DatagramSocket(port);

        byte[] buffer = new byte[1024];
        DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
        socket.receive(packet);

        String message = new String(packet.getData(), 0, packet.getLength());
```

```
        System.out.println("Received message: " + message);
    }
}
```

Pada sisi server, program membaca input dari pengguna menggunakan BufferedReader dan mengirimkan pesan ke alamat IP localhost dengan port 12345 menggunakan DatagramSocket dan DatagramPacket. Pada sisi client, program menerima pesan yang dikirimkan ke port 12345 menggunakan DatagramSocket dan DatagramPacket, lalu menampilkan pesan tersebut ke layar.

Namun, perlu diperhatikan bahwa contoh ini tidak memiliki mekanisme verifikasi input yang diterima dari sisi aplikasi client. Untuk melakukan verifikasi input, bisa dilakukan validasi data pada sisi server sebelum mengirimkan pesan.

Berikut ini contoh implementasi di sisi server (pengirim) dan sisi client (penerima) dengan tambahan mekanisme verifikasi input:

Server (Pengirim):

Server.java

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

public class Server {
   public static void main(String[] args) {
      try {
         // Buka server socket pada port 8888
         ServerSocket serverSocket = new ServerSocket(8888);

         // Tunggu koneksi dari client
         System.out.println("Menunggu koneksi dari client...");
         Socket socket = serverSocket.accept();
         System.out.println("Koneksi diterima dari client " + socket.getInetAddress().getHostName());

         // Buat output stream untuk mengirim data ke client
         OutputStream outputStream = socket.getOutputStream();
         DataOutputStream dataOutputStream = new DataOutputStream(outputStream);

         // Kirim data ke client
         String pesan = "Ini adalah pesan dari server.";
         String verifikasi = "1234567890"; // contoh verifikasi
         dataOutputStream.writeUTF(pesan + verifikasi); // tambahkan verifikasi ke pesan

         // Tutup output stream dan socket
         dataOutputStream.close();
         socket.close();
      } catch (IOException e) {
```

```
            e.printStackTrace();
        }
    }
}
```

Client.java

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

public class Client {
    public static void main(String[] args) {
        try {
            // Buka socket dan koneksi ke server
            Socket socket = new Socket("localhost", 8888);
            System.out.println("Terhubung ke server " + socket.getInetAddress().getHostName());

            // Buat input stream untuk membaca data dari server
            InputStream inputStream = socket.getInputStream();
            DataInputStream dataInputStream = new DataInputStream(inputStream);

            // Baca data dari server
            String pesan = dataInputStream.readUTF();
            String verifikasi = pesan.substring(pesan.length() - 10); // ambil verifikasi dari pesan
            pesan = pesan.substring(0, pesan.length() - 10); // hapus verifikasi dari pesan

            // Verifikasi input dari server
            if (verifikasi.equals("1234567890")) {
                System.out.println("Pesan dari server: " + pesan);
            } else {
                System.out.println("Pesan dari server tidak valid.");
            }

            // Tutup input stream dan socket
            dataInputStream.close();
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Pada contoh di atas, mekanisme verifikasi input dilakukan dengan cara menambahkan kode verifikasi ke pesan yang dikirimkan dari server, dan mengambil kode verifikasi tersebut dari pesan yang diterima oleh client. Kode verifikasi dapat berupa hash, checksum, atau kode acak lainnya yang

dihasilkan oleh server dan dikirimkan bersama-sama dengan pesan. Setelah kode verifikasi diterima oleh client, client dapat melakukan verifikasi dengan membandingkan kode verifikasi tersebut dengan kode verifikasi yang diharapkan. Jika kedua kode verifikasi tersebut sama, maka pesan dianggap valid dan dapat diproses oleh client. Jika kedua kode verifikasi tersebut tidak sama, maka pesan dianggap tidak valid dan harus diabaikan.

Validasi input server:

Validasi input pada sisi server dapat dilakukan dengan beberapa cara, tergantung dari jenis data yang diterima dan aturan validasi yang ditetapkan. Beberapa contoh metode validasi input pada sisi server adalah sebagai berikut:

1. Validasi tipe data: Memastikan bahwa input yang diterima memiliki tipe data yang sesuai dengan yang diharapkan. Misalnya, jika server mengharapkan input berupa bilangan bulat, maka harus dilakukan validasi apakah input tersebut memang berupa bilangan bulat atau tidak.
2. Validasi panjang data: Memastikan bahwa input yang diterima memiliki panjang yang sesuai dengan yang diharapkan. Misalnya, jika server mengharapkan input berupa teks dengan panjang maksimal 100 karakter, maka harus dilakukan validasi apakah input tersebut tidak melebihi 100 karakter.
3. Validasi range data: Memastikan bahwa input yang diterima memiliki nilai yang berada dalam range yang diharapkan. Misalnya, jika server mengharapkan input berupa bilangan bulat antara 1 sampai 100, maka harus dilakukan validasi apakah input tersebut memiliki nilai yang berada dalam range tersebut atau tidak.
4. Validasi format data: Memastikan bahwa input yang diterima memiliki format yang sesuai dengan yang diharapkan. Misalnya, jika server mengharapkan input berupa alamat email, maka harus dilakukan validasi apakah input tersebut memiliki format email yang benar.

Contoh implementasi validasi input pada sisi server dalam bahasa Java:

Server.java

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Server {
    public static void main(String[] args) {
        // Menerima input dari client
        String input = "10";

        // Validasi input sebagai bilangan bulat positif
        try {
            int number = Integer.parseInt(input);
            if (number < 0) {
                throw new NumberFormatException();
            }
            System.out.println("Input valid: " + number);
        } catch (NumberFormatException e) {
            System.out.println("Input tidak valid: harus berupa bilangan bulat positif.");
        }
```

```
        // Validasi input sebagai alamat email
        String email = "example.com";
        Pattern pattern = Pattern.compile("[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-
]+\\.[a-zA-Z]{2,}");
        Matcher matcher = pattern.matcher(email);
        if (matcher.matches()) {
            System.out.println("Input valid: " + email);
        } else {
            System.out.println("Input tidak valid: harus berupa alamat email yang
benar.");
        }
    }
}
```

Dalam contoh di atas, terdapat dua contoh validasi input pada sisi server. Yang pertama adalah validasi input sebagai bilangan bulat positif, dan yang kedua adalah validasi input sebagai alamat email. Dalam validasi input sebagai bilangan bulat positif, dilakukan pengecekan apakah input tersebut dapat di-parse sebagai bilangan bulat positif atau tidak. Sedangkan dalam validasi input sebagai alamat email, dilakukan pengecekan apakah input tersebut memiliki format email yang benar menggunakan regular expression.

Verifikasi keaslian

Verifikasi keaslian (authentication) merupakan aspek kritis dalam keamanan informasi dan sistem komputer. Ini mengacu pada proses memastikan bahwa entitas yang terlibat dalam komunikasi atau interaksi adalah entitas yang seharusnya dan tidak ada yang mencoba menyamar atau memberikan identitas palsu. Dalam konteks transfer file atau data antar komponen, verifikasi keaslian sangat penting dengan beberapa alasan:

1. Mencegah Identitas Palsu : Verifikasi keaslian membantu mencegah penyamaran atau identitas palsu. Tanpa verifikasi keaslian, entitas dapat berusaha untuk memasuki atau berkomunikasi dengan sistem dengan identitas palsu, membahayakan keamanan dan integritas sistem.
2. Proteksi terhadap Serangan Man-in-the-Middle (MitM): Dengan mengonfirmasi keaslian entitas, kita dapat mencegah serangan Man-in-the-Middle di mana seorang penyerang mencoba mengintersepsi dan memanipulasi komunikasi antara dua entitas. Verifikasi keaslian membantu memastikan bahwa komunikasi hanya terjadi antara entitas yang sah.
3. Kontrol Akses: Verifikasi keaslian memungkinkan untuk mengontrol akses ke sistem atau sumber daya berdasarkan identitas entitas yang meminta akses. Ini memastikan bahwa hanya entitas yang sah yang diberikan izin untuk mengakses informasi atau sumber daya tertentu.
4. Integritas Data: Dalam konteks transfer file atau data, verifikasi keaslian juga berkontribusi pada memastikan integritas data. Jika pengirim atau penerima file tidak dapat diandalkan, data mungkin dapat dimanipulasi selama transit, mengakibatkan ketidakintegritasan dan potensi risiko keamanan.

5. Kepatuhan dan Audit:Dalam banyak keadaan, organisasi perlu mematuhi regulasi atau standar keamanan tertentu. Verifikasi keaslian membantu dalam mencapai dan mematuhi persyaratan ini, dan juga memudahkan audit keamanan untuk melacak dan memeriksa aktivitas entitas yang terlibat.
6. Pemeliharaan Reputasi dan Kepercayaan: Verifikasi keaslian juga penting untuk memelihara reputasi dan kepercayaan. Dalam dunia bisnis dan layanan online, konsumen dan mitra bisnis perlu yakin bahwa mereka berurusan dengan entitas yang sebenarnya dan dapat diandalkan.
7. Perlindungan Terhadap Ancaman Keamanan:Dengan memverifikasi keaslian entitas, kita dapat mengurangi risiko terhadap berbagai jenis ancaman keamanan seperti serangan phishing, serangan identitas palsu, dan penggunaan akses yang tidak sah.

Dengan demikian, verifikasi keaslian bukan hanya langkah keamanan tambahan, melainkan fondasi yang krusial untuk menciptakan ekosistem komputasi yang aman dan andal. Implementasi yang tepat dari verifikasi keaslian dapat mengurangi risiko keamanan, melindungi integritas data, dan memberikan dasar kepercayaan yang diperlukan dalam lingkungan digital.

Untuk memastikan bahwa pengiriman file tersebut seutuhnya asli dan tidak mengalami perubahan, kita dapat menggunakan metode yang disebut dengan hash function. Hash function menghasilkan nilai hash (checksum) unik berdasarkan konten file. Jika isi file berubah, nilai hashnya juga akan berubah. Dengan membandingkan nilai hash file yang diterima dengan nilai hash yang dihasilkan pada sisi pengirim, kamu dapat memverifikasi keaslian dan integritas file.

Berikut adalah langkah-langkah umum untuk melakukan verifikasi ini:

1. Membuat Hash pada Sisi Pengirim, pengirim menghasilkan nilai hash dari file menggunakan hash function seperti MD5, SHA-256, atau SHA-3. Contoh dalam Java menggunakan SHA-256:

```
    MessageDigest digest = MessageDigest.getInstance("SHA-256");
    byte[] hash = digest.digest(fileContent);
    ```
```

2. Mengirim File bersama dengan Nilai Hash, pengirim mengirimkan file dan nilai hash ke penerima.
3. Menerima File dan Nilai Hash pada Sisi Penerima, penerima menerima file dan nilai hash.
4. Membuat Hash pada Sisi Penerima:Penerima menggunakan hash function yang sama untuk menghasilkan nilai hash dari file yang diterima. Contoh dalam Java:

```
  MessageDigest digest = MessageDigest.getInstance("SHA-256");
  byte[] receivedHash = digest.digest(receivedFileContent);
    ```
```

5. Membandingkan Nilai Hash, penerima membandingkan nilai hash yang diterima dengan nilai hash yang dikirim oleh pengirim. Jika nilai hash sesuai, file dianggap tidak berubah dan dapat dianggap asli.

Penting untuk menggunakan algoritma hash yang aman dan sulit dipecahkan (collision-resistant) untuk memastikan keamanan verifikasi integritas file. MD5, contohnya, sekarang dianggap rentan terhadap serangan collision dan tidak lagi direkomendasikan untuk keamanan tinggi. Perhatikan bahwa meskipun verifikasi hash dapat membantu memastikan keaslian dan integritas file,

ini tidak menjamin keamanan keseluruhan sistem. Dalam konteks keamanan yang lebih tinggi, pertimbangkan juga penggunaan tanda tangan digital atau metode keamanan yang lebih canggih.

Berikut ini contoh sederhana implementasi verifikasi integritas file menggunakan nilai hash pada sisi server (pengirim) dan client (penerima) menggunakan bahasa pemrograman Java.

Sisi Server (Pengirim):

```
1. import java.io.*;
2. import java.nio.file.*;
3. import java.security.*;
4.
5. public class FileSender {
6.
7.     public static void main(String[] args) {
8.         try {
9.             // Path file yang akan dikirim
10.            Path filePath = Paths.get("path/to/your/file.txt");
11.            byte[] fileContent = Files.readAllBytes(filePath);
12.
13.            // Menggunakan SHA-256 untuk menghitung nilai hash
14.            MessageDigest digest = MessageDigest.getInstance("SHA-256");
15.            byte[] hash = digest.digest(fileContent);
16.
17.            // Mengirim file dan nilai hash ke client
18.            sendFileAndHash(fileContent, hash);
19.        } catch (IOException | NoSuchAlgorithmException e) {
20.            e.printStackTrace();
21.        }
22.    }
23.
24.    private static void sendFileAndHash(byte[] fileContent, byte[] hash) {
25.        // Implementasi pengiriman file dan hash ke client
26.        // Contoh: Gunakan socket programming atau mekanisme pengiriman lainnya
27.        // ...
28.    }
29. }
30. ```
31.
```

Sisi Client (Penerima):

```
1. import java.io.*;
2. import java.nio.file.*;
3. import java.security.*;
4.
5. public class FileReceiver {
6.
7.     public static void main(String[] args) {
8.         try {
9.             // Menerima file dan hash dari server
10.            byte[] receivedFileContent = receiveFile();
11.            byte[] receivedHash = receiveHash();
12.
13.            // Menggunakan SHA-256 untuk menghitung nilai hash dari file yang diterima
14.            MessageDigest digest = MessageDigest.getInstance("SHA-256");
15.            byte[] calculatedHash = digest.digest(receivedFileContent);
16.
```

```
            // Membandingkan nilai hash
            if (MessageDigest.isEqual(calculatedHash, receivedHash)) {
                System.out.println("File integritas terjamin. File asli.");
            } else {
                System.out.println("Perubahan terdeteksi. File tidak dapat diandalkan.");
            }
        } catch (IOException | NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }

    private static byte[] receiveFile() throws IOException {
        // Implementasi penerimaan file dari server
        // Contoh: Gunakan socket programming atau mekanisme penerimaan lainnya
        // ...
        return new byte[0];
    }

    private static byte[] receiveHash() throws IOException {
        // Implementasi penerimaan nilai hash dari server
        // Contoh: Gunakan socket programming atau mekanisme penerimaan lainnya
        // ...
        return new byte[0];
    }
}
```

Pastikan untuk mengganti "path/to/your/file.txt" dengan path file yang sesuai pada sisi server. Implementasi pengiriman dan penerimaan file serta hash dapat disesuaikan dengan kebutuhan menggunakan metode seperti socket programming atau protokol komunikasi lainnya.

Latihan :

Silahkan coba program dibawah ini dan sesuaikan dengan kebutuhan anda, misalkan jenis file yang dikirim dan sebagaina :

Sisi Pengirim (Sender):

```
import java.io.*;
import java.nio.file.*;
import java.security.*;
import java.math.BigInteger;

public class SecureFileSender {

    public static void main(String[] args) {
        try {
            // Path file yang akan dikirim
            Path filePath = Paths.get("path/to/your/file.txt");
            byte[] fileContent = Files.readAllBytes(filePath);

            // Menggunakan MD5 untuk menghitung nilai hash
            String hash = calculateMD5(fileContent);

            // Mengirim file dan nilai hash ke client
            sendFileAndHash(fileContent, hash);
        } catch (IOException | NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }

    private static String calculateMD5(byte[] fileContent) throws NoSuchAlgorithmException {
        MessageDigest md = MessageDigest.getInstance("MD5");
```

```
        byte[] hashBytes = md.digest(fileContent);

        // Convert ke format hexadecimal
        BigInteger number = new BigInteger(1, hashBytes);
        String hash = number.toString(16);

        // Padding nol jika perlu
        while (hash.length() < 32) {
            hash = "0" + hash;
        }

        return hash;
    }

    private static void sendFileAndHash(byte[] fileContent, String hash) {
        // Implementasi pengiriman file dan nilai hash ke client
        // Contoh: Gunakan socket programming atau mekanisme pengiriman lainnya
        // ...
    }
}
```

Sisi Penerima (Receiver):

```
import java.io.*;
import java.nio.file.*;
import java.security.*;
import java.math.BigInteger;

public class SecureFileReceiver {

    public static void main(String[] args) {
        try {
            // Menerima file dan nilai hash dari server
            byte[] receivedFileContent = receiveFile();
            String receivedHash = receiveHash();

            // Menggunakan MD5 untuk menghitung nilai hash dari file yang diterima
            String calculatedHash = calculateMD5(receivedFileContent);

            // Membandingkan nilai hash
            if (calculatedHash.equals(receivedHash)) {
                System.out.println("File integritas terjamin. File asli.");
            } else {
                System.out.println("Perubahan terdeteksi. File tidak dapat diandalkan.");
            }
        } catch (IOException | NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }

    private static byte[] receiveFile() throws IOException {
        // Implementasi penerimaan file dari server
        // Contoh: Gunakan socket programming atau mekanisme penerimaan lainnya
        // ...
        return new byte[0];
    }

    private static String receiveHash() throws IOException {
        // Implementasi penerimaan nilai hash dari server
        // Contoh: Gunakan socket programming atau mekanisme penerimaan lainnya
        // ...
        return "";
    }

    private static String calculateMD5(byte[] fileContent) throws NoSuchAlgorithmException {
```

```
43.         MessageDigest md = MessageDigest.getInstance("MD5");
44.         byte[] hashBytes = md.digest(fileContent);
45.
46.         // Convert ke format hexadecimal
47.         BigInteger number = new BigInteger(1, hashBytes);
48.         String hash = number.toString(16);
49.
50.         // Padding nol jika perlu
51.         while (hash.length() < 32) {
52.             hash = "0" + hash;
53.         }
54.
55.         return hash;
56.     }
57. }
58. ```
59.
```

Latihan diatas coba jalankan kedua sisinya, dan amati hasilnya

Tugas Tumah :

Buat program untuk mengirim file dan untuk memverifikasi file yang dikirim apakah masih asli atau sudah ada perubahan dengan menggunakan BLAKE2 atau SHA3.