Security and Privacy Resources using Distributed programming

By: Humaiz Javed

# **ABSTRACT**

This project presents a secure server-client architecture designed to enhance the security and privacy of valuable resources within a networked environment. The architecture employs a multithreaded server, accommodating multiple clients and facilitating secure communication channels. The server is equipped with mechanisms for authenticating a designated master node, responsible for managing resource access requests.

Clients are assigned unique identifiers, and the server generates and manages distinct passwords for each client to access resources securely. The project introduces a two-step authentication process for the master node, ensuring an additional layer of security. Once authenticated, the master node can generate, distribute, and manage passwords for individual clients, providing controlled access to specific resources.

The project's significance lies in its ability to establish a secure communication framework, preventing unauthorized access and ensuring the privacy of sensitive resources. The implementation addresses security concerns prevalent in contemporary networked systems, contributing to the development of robust solutions for safeguarding digital assets in various applications.

# **INTRODUCTION**

# In the era of digital transformation, ensuring the security and privacy of sensitive resources is paramount. As organizations increasingly rely on interconnected systems and networks, safeguarding critical information from unauthorized access becomes a challenging yet essential task. This project aims to address the security concerns associated with resource access by implementing a robust and scalable server-client architecture. The focus is on providing a secure communication channel while efficiently managing client authentication and resource authorization.

# **PROBLEM STATEMENT:**

In modern networked environments, the security and privacy of valuable resources face constant threats from unauthorized access and data breaches. Conventional server-client architectures often lack robust security mechanisms, leaving sensitive information vulnerable to exploitation. The challenge lies in developing a secure and scalable solution that effectively manages client authentication, ensures secure communication, and controls resource access in a dynamic networked environment.

## **METHODOLOGY:**

**The proposed solution adopts a multithreaded server-client architecture to handle concurrent connections securely. Clients are assigned unique identifiers, and the master node authentication process employs a two-step verification mechanism to enhance security. Passwords are generated for each client, providing a controlled means of resource access. The server employs encryption for communication, ensuring data integrity and confidentiality.**

**The methodology includes the implementation of a password management system, allowing the master node to generate, distribute, and validate passwords for clients. Resource requests are processed securely, and non-master node clients receive restricted communication capabilities to prevent unauthorized actions.**

# **PROJECT SCOPE:**

The scope of this project encompasses the development of a secure server-client architecture designed to address security and privacy concerns in resource access within networked environments. The primary focus is on creating a scalable and robust system that ensures confidentiality, integrity, and controlled access to sensitive resources.

* **In-Scope:**

Secure Communication: Implement a multithreaded server-client model with encrypted communication channels to prevent eavesdropping and tampering.

Client Authentication: Develop a two-step authentication process for the master node to enhance security and restrict unauthorized access.

Password Management: Implement a password generation and distribution system, allowing the master node to assign unique passwords to clients for controlled resource access.

Resource Access Control: Enable the server to manage and validate resource access requests, ensuring that clients can only access designated resources.

Error Handling: Implement robust error handling mechanisms to address potential issues during communication and authentication processes.

* **Out-of-Scope:**

Advanced Encryption Standards: While encryption is a crucial aspect, the project does not aim to implement the most advanced encryption standards. Future developments may explore more sophisticated encryption techniques.

Intrusion Detection: Real-time intrusion detection mechanisms are beyond the current scope. Future iterations may include proactive security measures for intrusion detection.

Adaptive Access Control: The initial implementation focuses on static access control. Future developments may explore adaptive access control based on dynamic network conditions.

User Interface (UI): The project primarily addresses backend functionality. The user interface for client interaction is considered out of scope.

# **SERVER CODE:**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Threading;

public class Server

{

private static Dictionary<int, List<string>> clientPasswords = new Dictionary<int, List<string>>();

private static int clientCount = 0;

private static object lockObject = new object();

private static bool isFirstClientConnected = false;

private static int masterNodeID = -1;

private static TcpClient masterNodeClient = null;

private static string masterNodePassword = "pndc";

private static bool isMasterNodeAuthenticated = false;

private static void ProcessClientRequests(object argument)

{

TcpClient client = (TcpClient)argument;

int clientid = Interlocked.Increment(ref clientCount);

try

{

NetworkStream stream = client.GetStream();

StreamReader reader = new StreamReader(stream);

StreamWriter writer = new StreamWriter(stream) { AutoFlush = true };

lock (lockObject)

{

if (!isFirstClientConnected)

{

isFirstClientConnected = true;

masterNodeID = clientid;

masterNodeClient = client; // Store the master node client for communication

Console.WriteLine($"Client {clientid} connected as Master Node.");

// Send a welcome message to the master node

writer.WriteLine($"Hello Master Node {clientid}! Please enter the password to proceed:");

}

else

{

Console.WriteLine($"Client {clientid} connected.");

// Send a message to non-master node clients

writer.WriteLine($"Client {clientid} You are not the Master Node. Communication is restricted.");

}

}

while (true)

{

string clientResponse = reader.ReadLine();

string[] clientResponseParts = clientResponse.Split(' ');

Console.WriteLine($"Response from Client {clientid}: {clientResponse}");

lock (lockObject)

{

if (clientid == masterNodeID)

{

// Process master node requests here

if (!isMasterNodeAuthenticated)

{

// Authenticate the master node with the entered password

string enteredPassword = clientResponseParts[0].ToLower();

if (enteredPassword.Equals(masterNodePassword, StringComparison.OrdinalIgnoreCase))

{

isMasterNodeAuthenticated = true;

Console.WriteLine($"Master Node {clientid} authenticated. You can now execute commands.");

// Send a welcome message to the master node

writer.WriteLine("Welcome, Master Node! You are now authenticated.");

}

else

{

Console.WriteLine($"Incorrect password entered by Master Node {clientid}.");

// Send an error message to the master node

writer.WriteLine("Incorrect password. Access denied.");

return; // Exit the thread for non-authenticated master nodes

}

}

if (clientResponseParts.Length >= 3 && clientResponseParts[1].ToLower() == "for" && clientResponseParts[2].ToLower() == "client")

{

int requestedClientID;

if (int.TryParse(clientResponseParts.Last(), out requestedClientID))

{

if (clientPasswords.ContainsKey(requestedClientID))

{

string storedPassword = clientPasswords[requestedClientID][0];

string enteredPassword = clientResponseParts[0].ToLower();

// Check if entered password matches the stored password

if (enteredPassword == storedPassword.ToLower())

{

// Send success message to the client

writer.WriteLine("Password matched. You can now enter new commands.");

}

else

{

// Send failure message to the client

writer.WriteLine("Incorrect password. Access denied.");

}

}

else

{

Console.WriteLine($"Password not generated for Client {requestedClientID}. Please generate a password first.");

}

}

else

{

Console.WriteLine($"Invalid request format from Master Node: {clientResponse}");

}

}

// Add more conditions for master node requests here

}

else

{

// Non-master node requests

if (clientResponse.ToLower() == "send password")

{

if (clientPasswords.ContainsKey(clientid))

{

// Send password to the requesting client

SendPasswordToClient(writer, clientid);

}

else

{

writer.WriteLine("Password not generated for this client. Please request a password first.");

}

}

// Add more conditions for non-master node requests here

}

}

// Send a response back to the client

writer.WriteLine($"Server received: {clientResponse}");

}

}

catch (IOException ex)

{

Console.WriteLine($"Problem with Client {clientid} communication: {ex.Message}. Exiting thread.");

}

finally

{

if (client != null)

{

client.Close();

}

}

}

private static void SendPasswordToMasterNode(StreamWriter writer, int clientid)

{

if (clientPasswords.ContainsKey(clientid))

{

List<string> passwords = clientPasswords[clientid];

writer.WriteLine($"Password for Client {clientid}: {string.Join(", ", passwords)}");

}

else

{

writer.WriteLine($"Password for Client {clientid} not generated yet. Please wait.");

}

}

private static void GeneratePasswords(int count, int length, int clientid)

{

List<string> passwords = new List<string>();

Random random = new Random();

for (int i = 0; i < count; i++)

{

string password;

do

{

// Generate a random alphanumeric password with the specified length

password = new string(Enumerable.Repeat("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789", length)

.Select(s => s[random.Next(s.Length)]).ToArray());

} while (passwords.Contains(password)); // Ensure uniqueness

passwords.Add(password);

}

// Assign the generated passwords to the respective client

clientPasswords[clientid] = passwords;

// Display generated passwords on the server's console

Console.WriteLine($"Client {clientid} Password Set: {string.Join(", ", passwords)}");

// Signal waiting threads that new passwords are available

Monitor.PulseAll(lockObject);

}

private static void ForwardRequestToMasterNode(int clientid, string request)

{

if (masterNodeClient != null)

{

NetworkStream stream = masterNodeClient.GetStream();

StreamWriter masterNodeWriter = new StreamWriter(stream) { AutoFlush = true };

// Forward the request to the master node

masterNodeWriter.WriteLine($"Client {clientid} requests: {request}");

}

else

{

Console.WriteLine("Master Node not available. Cannot forward request.");

}

}

private static void SendPasswordToClient(StreamWriter writer, int clientid)

{

if (clientPasswords.ContainsKey(clientid))

{

List<string> passwords = clientPasswords[clientid];

writer.WriteLine($"Password for Client {clientid}: {string.Join(", ", passwords)}");

}

else

{

writer.WriteLine($"Password for Client {clientid} not generated yet. Please wait.");

}

}

public static void Main()

{

TcpListener listener = null;

try

{

listener = new TcpListener(IPAddress.Parse("127.0.0.1"), 8080);

listener.Start();

Console.WriteLine("Server is waiting for the clients...");

while (true)

{

TcpClient manager = listener.AcceptTcpClient();

Thread t = new Thread(ProcessClientRequests);

t.Start(manager);

}

}

catch (Exception e)

{

Console.WriteLine(e);

}

finally

{

if (listener != null)

{

listener.Stop();

}

}

}

}

**CLIENT OUTPUT:**using System;

using System.IO;

using System.Net.Sockets;

public class Client

{

private static int clientPort = 8080;

public static void Main()

{

try

{

TcpClient client = new TcpClient("127.0.0.1", clientPort);

NetworkStream stream = client.GetStream();

StreamReader reader = new StreamReader(stream);

StreamWriter writer = new StreamWriter(stream) { AutoFlush = true };

string welcomeMessage = reader.ReadLine();

Console.WriteLine($"Server: {welcomeMessage}");

bool waitingForPassword = false;

int targetClientId = -1;

while (true)

{

Console.Write("Enter a command: ");

string userCommand = Console.ReadLine();

writer.WriteLine(userCommand);

if (waitingForPassword)

{

Console.Write("Enter password: ");

string enteredPassword = Console.ReadLine();

// Send the entered password and target client ID to the server

writer.WriteLine($"{enteredPassword} {targetClientId}");

// Receive the server response after entering the password

string response = reader.ReadLine();

if (response == "Password matched. You can now enter new commands.")

{

Console.Clear(); // Clear the console screen

waitingForPassword = false;

// Continue the loop to receive further commands

continue;

}

else

{

Console.WriteLine("Incorrect password. Exiting client.");

break;

}

}

else

{

// Handle other server responses

string response = reader.ReadLine();

Console.WriteLine($"Server response: {response}");

if (userCommand.ToLower() == "ok")

{

Console.Clear(); // Clear the console screen

waitingForPassword = true;

Console.Write("Enter target client ID: ");

if (int.TryParse(Console.ReadLine(), out targetClientId))

{

// Continue the loop to receive password

continue;

}

else

{

Console.WriteLine("Invalid target client ID. Exiting client.");

break;

}

}

}

}

}

catch (IOException ex)

{

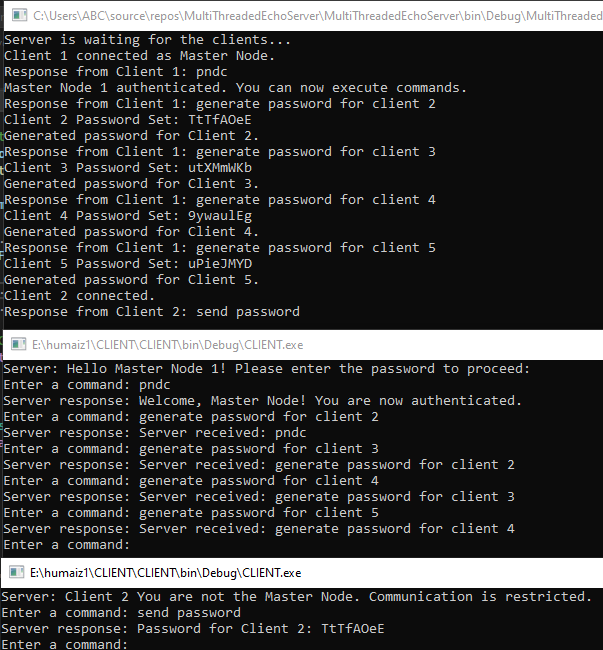
Console.WriteLine($"Problem with server communication: {ex.Message}. Exiting client.");

}

}

}

# **OUTPUT:**



## **FUTURE DEVELOPMENT**

# Future development efforts will focus on expanding the authentication and authorization mechanisms. Integration of advanced encryption techniques and the implementation of secure protocols will be explored to further enhance the project's security features. Additionally, the system can be extended to include features such as secure file transfer, intrusion detection, and adaptive access control based on real-time network conditions. Continuous research into emerging security threats will guide the integration of proactive security measures.

# **CONCLUSION**

In conclusion, this project presents an effective solution to the security and privacy challenges associated with resource access in networked environments. The implemented server-client architecture, with its emphasis on secure communication, client authentication, and controlled resource access, provides a foundation for addressing contemporary security concerns. As digital landscapes evolve, this project serves as a crucial step towards building resilient and adaptive systems that prioritize the confidentiality and integrity of sensitive resources. Future developments will contribute to advancing the project's capabilities and ensuring its relevance in the ever-changing landscape of cybersecurity.