**Communication security!**

**Lab Overview:** Implement secure communication between server and client, review encryption and authentication concepts.

**Step1. Analysis of the simple server-client communication code given for the problem in Step 2:**

* Template Code – Server (Soldier)

|  |
| --- |
| using System;  using System.Net;  using System.Net.Sockets;  using System.Text;  class Server  {  static void Main()  {  // ① Initialize server socket  TcpListener server = new TcpListener(IPAddress.Any, 5000);  server.Start();  Console.WriteLine("Waiting for commander’s instructions...");  while (true)  {  // ② Accept client connection  TcpClient client = server.AcceptTcpClient();  Console.WriteLine("Commander connected!");  // ③ Receive data  NetworkStream stream = client.GetStream();  byte[] buffer = new byte[256];  int bytesRead = stream.Read(buffer, 0, buffer.Length);  string receivedData =  Encoding.UTF8.GetString(buffer, 0, bytesRead);  Console.WriteLine($"Data received from commander: {receivedData}");  // ④ Send response  string response =  "I need to sleep at dawn. I will attack after lunch.";  buffer = Encoding.UTF8.GetBytes(response);  stream.Write(buffer, 0, buffer.Length);  client.Close();  }  }  } |

* Template Code – Client (Commander)

|  |
| --- |
| using System;  using System.Net.Sockets;  using System.Text;  class Client  {  static void Main()  {  // ① Server connection  TcpClient client = new TcpClient("127.0.0.1", 5000);  Console.WriteLine("Soldier connected!");  // ② Transfer data  NetworkStream stream = client.GetStream();  string message = "Attack at dawn!";  byte[] buffer = Encoding.UTF8.GetBytes(message);  stream.Write(buffer, 0, buffer.Length);  Console.WriteLine("Completed sending message to soldier!");  // ③ Receive response  buffer = new byte[256];  int bytesRead = stream.Read(buffer, 0, buffer.Length);  string response = Encoding.UTF8.GetString(buffer, 0, bytesRead);  Console.WriteLine($"Soldier response: {response}");  client.Close();  }  } |

**Step2. [5pt] Implementing simple data encryption:**

* Modify the server and client in Step 1 so that the following two functions (encryption and decryption) are created on both sides and both sides transmit and decrypt encrypted messages to verify the messages when exchanging messages.
* Encryption function

|  |
| --- |
| +Function signature: public static string SimpleEncrypt(string input);  +Purpose: Take a string as input, add a specific value to each character, and return an encrypted string  +Algorithm:  - Convert the input string to a character array.  - Iterate over each character in the array:  - Increase the ASCII code value of the current character by 10.  - Convert the modified character array to a string and return it. |

* Decryption function contents

|  |
| --- |
| + Function signature: public static string SimpleDecrypt(string input);  +Purpose: Take a string as input, subtract a specific value to each character, and return an decrypted string  +Algorithm:  - Convert the input string to a character array.  - Iterate over each character in the array:  - Decrease the ASCII code value of the current character by 10.  - Convert the modified character array to a string and return it. |

* output:

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**Step3. Let's learn briefly about Hash and analyze the given server-client (malicious) code.**

* Let’s implement data integrity using hashing to ensure that data is not damaged or tampered with during transmission. If a malicious client sends manipulated data to the server, the server determines whether it has been manipulated through hashing.
* **Hash**: A function that converts input data into a unique value (hash value) of fixed size. It is mainly used for efficiently storing, searching, and comparing data. (Used for data integrity checks, password storage, digital signatures and authentication, blockchain, etc.)
  1. **Representative hash algorithms** 
     1. MD5: Fast but has low security due to high collision probability.
     2. SHA-1: Better security, but currently recommended as a more secure algorithm.
     3. SHA-2 (SHA-256, etc.): Suitable for modern security standards.
     4. SHA-3: Next-generation hash standard providing high security.
  2. Example result of hash algorithm
     1. Input data: *hello*
     2. SHA-256 Hash value: 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
* Template Code - Server

|  |
| --- |
| using System;  using System.Net;  using System.Net.Sockets;  using System.Text;  using System.Security.Cryptography;  class Server  {  public static string ComputeHash(string data)  {  using (SHA256 sha256 = SHA256.Create())  {  byte[] bytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(data));  return Convert.ToBase64String(bytes);  }  }  static void Main()  {  TcpListener server = new TcpListener(IPAddress.Any, 5000);  server.Start();  Console.WriteLine("Server started!");  while (true)  {  TcpClient client = server.AcceptTcpClient();  Console.WriteLine("Client connected!");  NetworkStream stream = client.GetStream();  // Deceiving data  byte[] buffer = new byte[256];  int bytesRead = stream.Read(buffer, 0, buffer.Length);  string receivedData =  Encoding.UTF8.GetString(buffer, 0, bytesRead);  // Separation of data and hash  string[] parts = receivedData.Split('|');  string data = parts[0];  string receivedHash = parts[1];  // Integrity check  string computedHash = ComputeHash(data);  Console.WriteLine($"Generated hash: {computedHash}");  if (computedHash == receivedHash)  {  Console.WriteLine($"Data integrity verified: {data}");  string response = " Data received normally";  buffer = Encoding.UTF8.GetBytes(response);  stream.Write(buffer, 0, buffer.Length);  }  else  {  Console.WriteLine("Data is corrupted or falsified!");  string response = " Data corrupted!";  buffer = Encoding.UTF8.GetBytes(response);  stream.Write(buffer, 0, buffer.Length);  }  client.Close();  }  }  } |

* Template Code – Client (Hacker Client)

|  |
| --- |
| using System;  using System.Net.Sockets;  using System.Text;  class HackerClient  {  static void Main()  {  TcpClient client = new TcpClient("127.0.0.1", 5000);  NetworkStream stream = client.GetStream();  // Data manipulation (for example, manipulating the number of items)  string manipulatedMessage = "Game data: items=999, location=(10,20)";  string fakeHash = "fakeHashValue";  Console.WriteLine($"Generated hash: {fakeHash}");  // Sending data with incorrect hash  string dataToSend = $"{manipulatedMessage}|{fakeHash}";  byte[] buffer = Encoding.UTF8.GetBytes(dataToSend);  stream.Write(buffer, 0, buffer.Length);  // Receive server response  buffer = new byte[256];  int bytesRead = stream.Read(buffer, 0, buffer.Length);  string response = Encoding.UTF8.GetString(buffer, 0, bytesRead);  Console.WriteLine($"Server response: {response}");  client.Close();  }  } |

* The hacker client manipulates the data but does not generate the correct hash. Therefore, if it sends a forged value to the server, the server will recognize it as an incorrect hash value.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

* [5pt] Modify the client side to work as follows (don't modify the server):

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

**Step4. Both AES encryption and SHA256 hashing are applied to prevent sniffing.**

* [5pt] Rewrite the server-client communication in Step 3 using Encrypt() and Decrypt() from the Helper class given below.

|  |
| --- |
| //Helper class  using System;  using System.IO;  using System.Security.Cryptography;  using System.Text;  namespace encryptionHelper  {  public class AesEncryptionHelper  {  private static readonly string Key = "1234567890123456";  // 16 byte encryption key  private static readonly string IV = "1234567890123456";  // 16 byte initialization vector  public static string Encrypt(string plainText)  {  using (Aes aes = Aes.Create())  {  aes.Key = Encoding.UTF8.GetBytes(Key);  aes.IV = Encoding.UTF8.GetBytes(IV);  using (var ms = new MemoryStream())  {  using (var cs = new CryptoStream (ms, aes.CreateEncryptor(), CryptoStreamMode.Write))  {  using (var sw = new StreamWriter(cs))  {  sw.Write(plainText);  }  }  return Convert.ToBase64String(ms.ToArray());  }  }  }  public static string Decrypt(string cipherText)  {  using (Aes aes = Aes.Create())  {  aes.Key = Encoding.UTF8.GetBytes(Key);  aes.IV = Encoding.UTF8.GetBytes(IV);  using (var ms = new MemoryStream (Convert.FromBase64String(cipherText)))  {  using (var cs = new CryptoStream (ms, aes.CreateDecryptor(), CryptoStreamMode.Read))  {  using (var sr = new StreamReader(cs))  {  return sr.ReadToEnd();  }  }  }  }  }  }  } |

* 1. Example of result value (no corruption):

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

-------------------------------------------------------------------------------------------------------------------

Submission Guidelines:

* During the lab time, check and leave after completing Step 2, Step 3 and Step 4
* Submission: One word (docx), pdf or Hangul (hwpx) file with the source code in each step
  + The first line of the submission file must state which problem was solved (if not, 0 points)
  + The source code must be in text format, not an image
  + Results: No need to submit

■