Implementing Secure Client/Server Applications Using Java

Javier Pastorino

1

Outline

- Case Study
- Sending Data over a Network – Basis
- Sockets in Java
  - Simple Client/Server
  - Messaging System Prototype
- SSL/TLS
- Secure Sockets in Java
  - Setting up Server Certificates and Client TrustStore
  - Secure Messaging System Prototype

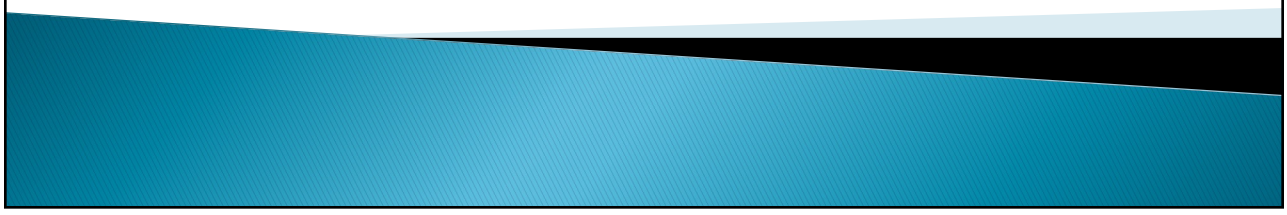
2

Case Study: Messaging Systems

- Messaging Systems allow different users to share messages over a communication channel.
  - This type of application has become very popular in the last few years.
  - E.g., Whatsapp, iMessage, and Messages by Google.
- A simple prototype can be built by implementing a Client/Server application.
  - The **server** will connect the clients and relay the messages sent between them.
  - The **client** will allow users to connect to the message network and interchange messages with other users.
- Users using these applications are concerned about the privacy of their messages.

3

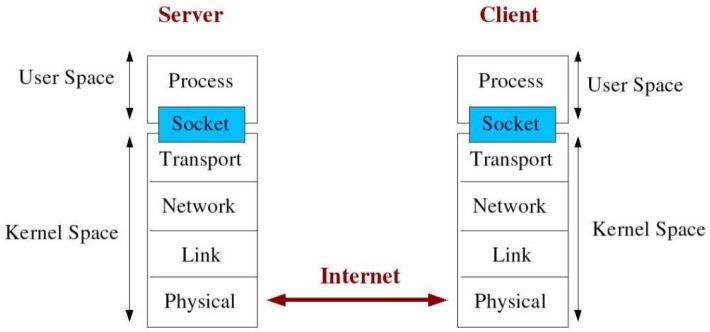
Sending Data over a Network



4

Socket

- **Is an interface between an application process and transport layer**
- The application process can send/receive messages to/from another application process, *local or remote,* via a socket



5

Socket
Fundamental Operations

- **`bind`**( )
  - Links a socket to a port on the local machine.
  - The port number is used by the kernel to match an incoming packet to a process
- **`listen`**( )
  - Waits for incoming connections.
- **`accept`**( )
  - Gets a pending connection from the listen()-ing socket.
  - Returns *a new socket* connected to the remote site
- **`connect`**( )
  - Connects the socket to a remote host.
- **`send()`**
  - Transmit a sequence of bytes from one socket to another.
- **`recv()`**
  - Receives a sequence of bytes from a connected socket.

6

7

Sockets in Java

- In Java, we have two classes to represent sockets.
- `import java.net.ServerSocket;`
  - Represent the server socket that will listen to connections
- `import java.net.Socket;`
  - Represent the client socket (the one that connects two points)
- `ServerSocket server = new ServerSocket(port, backlog);`
  - Creates a Server Socket Listening in the port with a queue length of the backlog
- `Socket client = server.accept();`
  - Waits for a client to connect.
  - Returns a Socket object representing the connection with the client.

8

Sockets in Java (cont'd)

Input and Output Streams

- Data Transfer in Java will be done using Data Streams
- The object class will depend on the data type to transfer

- **Output Stream** (Strings): *Used to send (write) data to the other point*

```
output = new PrintWriter(client.getOutputStream(),true);
output.println(message);
```

- **Input Stream** (Strings): *Used to receive (read) data from the other source*

```
input = new BufferedReader(new InputStreamReader(client.getInputStream()));
msg   = input.readLine();
```

Implementing Secure Client/Server Applications Using Java J. Pastorino 9

9

Simple Server - Java

```
import java.net.ServerSocket; import java.net.Socket;

public class SimpleServer {
  public static void main(String[] args) {
    ServerSocket socketServer = new ServerSocket(9888, 5);
    while(true){
       Socket connection = socketServer.accept();

       PrintWriter    output = new PrintWriter(connection.getOutputStream(),true);
       BufferedReader input  = new BufferedReader(new InputStreamReader(connection.getInputStream()));

       Thread.sleep(3000); //wait for 3 seconds (do processing.)

       output.println("Connected to Java server.");

       output.close();
       input.close();
       connection.close();
    }
  }
}
```

Implementing Secure Client/Server Applications Using Java J. Pastorino 10

10

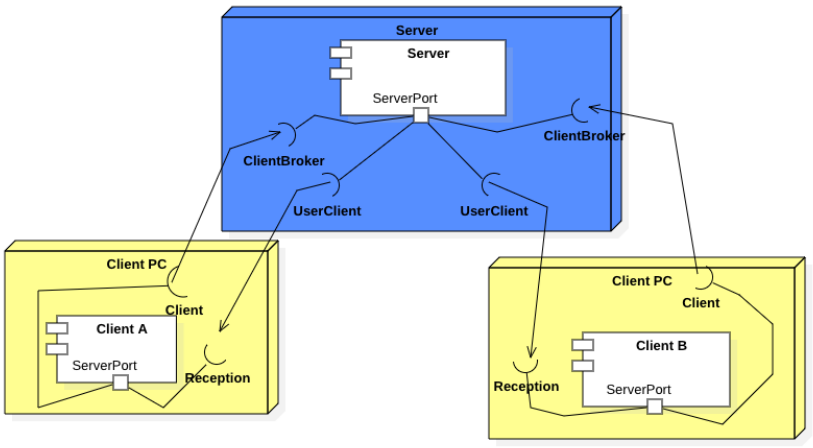
Simple Client – Java

```java
import java.net.Socket;

public class SimpleClient {
    public static void main(String[] args){

        Socket client = new Socket("localhost", 9888);

        PrintWriter    output = new PrintWriter(client.getOutputStream(),true);
        BufferedReader input  = new BufferedReader(new InputStreamReader(client.getInputStream()));

        String response = input.readLine();

        System.out.println(response);

        output.close();
        input.close();
        client.close();
    }
}
```

Implementing Secure Client/Server Applications Using Java J. Pastorino 11

11

Basic Messaging System



Implementing Secure Client/Server Applications Using Java J. Pastorino 12

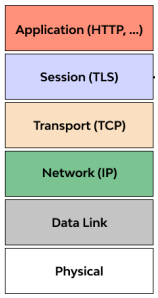
12

SSL/TLS

13

SSL/TLS

- Secure Socket Layer (SSL) and Transport Layer Security (TLS) are cryptographic protocols that provide authentication and data encryption between servers, machines, and applications operating over a network.
- Both SSL and TLS protocols are conceptually similar, but the key difference is how each achieves connection encryption.
  - Both refer to the handshake process between the client and server.
  - TLS 1.1 → 2006 TLS 1.2 → 2008 TLS 1.3 → 2018
  - Both SSL 2.0 and 3.0 are deprecated
- Secure = encryption/**C**onfidentiality + **I**ntegrity + **A**uthentication

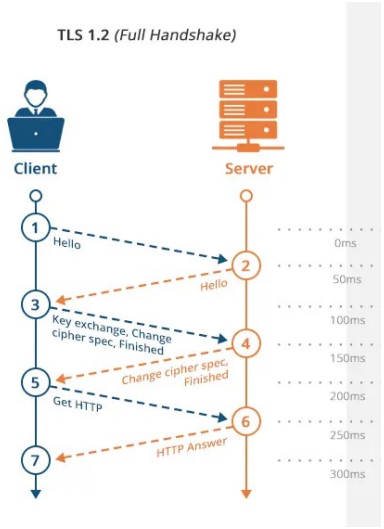


Implementing Secure Client/Server Applications Using Java J. Pastorino 14

14

TLS Handshake

- Handshake's goals are to authenticate the server and establish the encryption keys to use.
- Through the exchange of messages, both the Client and Server negotiate protocols and share configuration parameters for communication.
  - Server authentication relies on PK.
  - Additional mechanisms to prevent other attacks, like replay attacks, are included within the protocol.



15

Secure Client/Server End-To-End

- The attacker runs a sniffer to capture traffic.
  - Communication is C/S encrypted.
- DNS cache poisoning? DHCP spoofing?
  - The client goes to the wrong server. → Server authentication by PK
- The attacker hijacks the connection and injects new traffic
  - Data receiver rejects due to failed integrity checks.
- Routing manipulation to eavesdrop on traffic and redirect to a different server?
  - Messages are encrypted and impersonation can be detected
- Man in The Middle
  - Encryption → can't read and can't inject
  - Can't even replay previous TLS handshakes

16

17

SSL Sockets in Java

- `import javax.net.ssl.SSLServerSocket;`
- `import javax.net.ssl.SSLSocket;`

- Factories are needed to get the SSL Sockets.
  - **SSLSocket**

```
sf = (SSLSocketFactory)SSLSocketFactory.getDefault();
sslSocket = (SSLSocket)sslFact.createSocket(ip, port);
```

  - **SSLServerSocket**

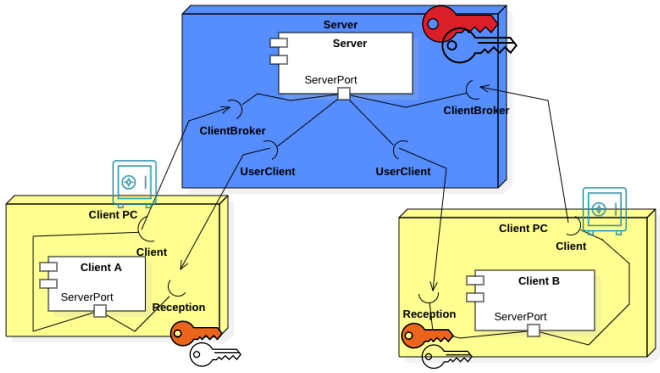
```
sslSrvF = (SSLServerSocketFactory)SSLServerSocketFactory.getDefault();
serverSocket = (SSLServerSocket)sslSrvF.createServerSocket(port, backlog);
```

Implementing Secure Client/Server Applications Using Java J. Pastorino 18

18

Basic Messaging System

- We need certificates for the server and a "trust store" for the clients.
- Self-signed certificates can be generated with **`keytool`** available with JDK.



19

SSL Sockets – Java Other Parameters

- Execution of SSL/TLS applications requires setting information about the certificates:
- Starting a Server:

```
java -Djavax.net.ssl.keyStore=keystore
     -Djavax.net.ssl.keyStorePassword=pass Server
```

- Starting a Client:

```
java -Djavax.net.ssl.trustStore=truststore
     -Djavax.net.ssl.trustStorePassword=pass Client
```


20

Summary

- TLS and *javax.net.ssl* package provide an easy way to protect our client/server communication.
- Next Steps:
  - **End-To-End encryption**. The server can still read and manipulate the messages a client sends to another client.
    - Hints: encrypt the message before leaving the server.
  - **Sender authentication**. Make sure the message is from who it says.
    - Hint: PK at the client level

21

References

- Oracle Security Developer's Guide – Chapter 8
- Java Development Kit API Reference
- https://www.globalsign.com/en/blog/ssl-vs-tls-difference
- Prototypes will be available at
  - https://github.com/jpastorino/CSCY_GestLecture

22

23