Secure Sockets

Module java.base
Package javax.net.ssl

TLS (SSL) Overview

- Protocols to enables secure communication between network endpoints
  - Transport Layer Security (TLS)
  - Secure Socket Layer (SSL) - deprecated!
- Provides C and I (from the CIA requirements)
  - Confidentiality - data sent is encrypted
  - Integrity - ensures data is not altered
  - Availability - TLS doesn’t really protect against Denial-of-Service
- TLS 1.0 and TLS 1.1 were to be deprecated in 2020

Converting EchoClient to SecureClient

EchoClient.java

```
    try {
        int portNumber = Integer.parseInt(args[1]);
        Socket echoSocket = new Socket(hostName, portNumber);
```

SecureClient.java

```
    SSLSocketFactory factory =
                (SSLSocketFactory)SSLSocketFactory.getDefault();

     try {
          int portNumber = Integer.parseInt(args[1]);
          SSLSocket echoSocket = (SSLSocket)factory.createSocket(
                      hostName, portNumber);
```

Converting from EchoServer

EchoServer.java

```
    try {
        int portNumber = Integer.parseInt(args[0]);
        ServerSocket serverSocket = new ServerSocket(portNumber);

        System.out.println("The server is listening at: " +

```

Converting to SecureServer

```
SSLServerSocketFactory factory =
            (SSLServerSocketFactory)SSLServerSocketFactory.getDefault();
 try {
      int portNumber = Integer.parseInt(args[0]);
      SSLServerSocket serverSocket =
                (SSLServerSocket)factory.createServerSocket(portNumber);

     System.out.println("The server is listening at: " +
            serverSocket.getInetAddress() + " on port " +
            serverSocket.getLocalPort());

     SSLSocket clientSocket = (SSLSocket)serverSocket.accept();
```

Generating a keystore/ truststore

- Caution: Avoid using self-signed keystores or certificates in production environments.

  ```
  $ keytool  -genkeypair  -keystore mystore -keyalg RSA
  ```

- Respond to the prompts

  - Please use password ‘password’ when doing the exercise on last slide

Run the server and client

```
$ java -cp build -Djavax.net.ssl.keyStore=./mystore  \
-Djavax.net.ssl.keyStorePassword=[your password]  \
SSServer 4444
```

```
$ java -cp build -Djavax.net.ssl.trustStore=./mystore  \
-Djavax.net.ssl.trustStorePassword=[your password]  \
SSClient localhost 4444
```

Lab Exercise

- Use the SimpleSocketDemo to create secure versions of the client and server
  - Copy SimpleSocketDemo to SecureSocketDemo
  - Modify the *.java files to use Secure Sockets
    - Be sure to update the import statements in all files
- Create a self-signed keystore with password “password”
- Verify that your secure client and server can connect and exchange messages