UNIT III

SOCKET FOR CLIENTS

* Data is transmitted across the internet in packets of finite size called datagrams.
* Each datagram contains a header and a payload.
* The header contains the address and port to which the packet is going, the address and port from which the packet came
* The payload contains data itself.
* Sockets allow the programmer to treat a network connection as just another stream onto which bytes can be written and from which bytes can be read.

Socket basics:

* A socket is a connection between two hosts. It can perform seven basic operations
* Connect to a remote machine
* Send data
* Receive data
* Close a connection
* Bind a port
* Listen for incoming data
* Accept connections from remote machines on the bound port.

Java socket class which is used by both clients and server, that correspond to the first four of these operations.

CLIENT SOCKET:

* The program creates a new socket with a constructor
* The socket attempts to connect to the remote host.
* Once the connection is established the local and remote hosts get input and output streams f rom the socket and use those streams to send data to each other.
* Some protocol such as HTTP 1.0 require the connection to be closed after each request is serviced. Others such as FTP allow multiple request to be processed in a single connection.

Investigating protocols with Telnet:

* Sockets communicate with well known Internet services such as HTTP, echo, and more.
* Telnet connect to a server, by typing different commands to it watch its responses.
* By default, telnet attempts to connect to port 23
* To connect to servers on different ports specify the port you want to connect to .
* SMTP is the protocol used to transfer email between servers or between email client.

The socket class:

* The java.net.socket class is java’s fundamental class for performing client side TCP operations.
* The TCP network connections such as URL, URL connection, Applet, JEditorPane all ultimately end up invoking the methods of this class
* The class uses native code to communicate with local TCP stack of the host operating system.
* TCP sockets are more or less reliable connections
* The interface that socket class provides to the programmer is streams.
* The actual reading and writing of data over the socket is accomplished via the familiar stream classes.

The Constructor:

* Public socket constructors are simple
* They let you specify the host and the port you want to connect to
* Host may be specified as an InetAddress or a string.
* Constructors also specify the local address and local port from which data will be sent.

Public socket(String host,int port) throws UnknownHostException,IOException.

* This constructor creates a TCP socket to the specified port on specified host and attempts to connect to remote host.

Server

Socket()

Socket()

bind()

listen()

establish connection

connect()

accept()

Send()

send()

recv()

\*call block \* call block

Wait for connection request

\*

recv()

Response

\* call block

Programs:

Find out which of the first 1024 ports seem to be hosting TCP servers on a specified host.

Import java.io.\*;

Import java.net.\*;

Public class lowportscanner{

Public static void main(String[] args)

String host=”local host”;

If(args.lengt>0){

Host=args[0];

}

For(int i=1;i<1024;i++)

{

Try{

Socket s=new socket(host,i);

System.out.println( “there is a server on port” + I “of” +host);

}

Catch(UnknownHostException ex){

System.err.println(ex);

Break;

}

Catch(IOException ex){

System.err.println(ex);

Break;

}

Catch (IOException ex){

}

}

}

This program helps you understand what your system is doing so you can find possible entrance points for attackers. You may also find rogue servers for e.g lowportscanner might tell u that theres a server on port 800 which on further investigation turns out to be an HTTP server

Public Socket(InetAddress host, int port) throws IOException

* This constructor creates a TCP socket to the specified port on the specified host and tries to connect

Try

{

InetAddress s=InetAddress.getByName([www.google.com](http://www.google.com));

Socket ssocket=new socket(s,80);

// send and receive data

}

Catch(UnknownHostException ex) {

System.err.println(ex);

}

Catch(IOException ex) {

System.err.println(ex);

}

Socket Address:

* Socket address class represents a connection endpoint.
* Java.net.SocketAddress class is an empty abstract class with no methods it has a default constructor.

Package java.net.\*;

Public abstract class SocketAddress {

Public SocketAddress() {}

}

* The main purpose of socket address class is to provide a transient socket connection information such as IPAddress and port that can be used to create a new socket.
* To this end the socket offers two methods that returns socket address objects

Public SocketAddress getRemoteSocketAddress()

Public SocketAddress getLocalSocketAddress()

* Both this method returns null if the socket is not connected.
* Socket address is needed to connect an unconnected socket via athe connect() method:Public void connect(SocketAddress endpoint) throws IOException

For e.g

First you might connect to yahoo then store its address

Socket socket= new Socket([www.yahoo.com](http://www.yahoo.com), 80);

socketAddress yahoo=Socket.getREmoteSocketAddress();

socket.close();

later it can be reconnected to yahoo using this address”

Socket =new Socket();

Socket.Connect(yahoo);

* Not all socket implementation can use the same subclasses of SocketAddress
* If an instance of wrong type is passed to connect(), it throws an IllegalArgumentException
* You can pass an int as second argument to specify the number of milliseconds to wait before the connection times out
* Public void connect(SocketAddress endpoint, int Timeout) throws IOException.
* The default 0 means wait for ever.

Socket for servers:

Servers are like receptionist who sit by the phone and wait for incoming calls. They don’t know who will call or when, only when the phone rings they have to pick up and talk to whoever is there. You cant program that behaviour with the socket class alone.

* In essence server sosckets job is to sit by the phone and wait for incoming TCP connection.

The Server Socket Class:

Step1: a new server socket is created on a particular port using serversocket() constructor

Step2: the server socket listens for incoming connection attempts on that port using its accept() method, accept() blocks until a client attempts to make connection at that point accept() returns a socket object connecting the client and the server.

Step3: depending on the type of server either the sockets get InputStream()mmethod getoutputStream() or both are called to get input and output streams that communicate with client

Step4: the server and client close according to the time agreement used in the protocol

Step5: the server the client or both close the connection;

Step6: the server returns the step2 and waits for the next connection

If step4 takes long or indefinite time traditional UNIX servers create a new process to handle each connection so that multiple clients can be serviced at the same time.

\*UNIX FTP server cant handle roughly more than 400 connection without slowing to a crawl.

Import java.io.\*;

Import java.net.\*;

Public class Myserver

{

Public static void main(String[] arg)

{

Try

{

ServerSocket ss=new ServerSocket(3333);

Socket s=ss.accept//establish connection

DataInputStream dis=new DataInputStream(s.getInputStream());

String str =(String)dis.readUTF();//embedding variable length and uses 8 bit code

System.out.println(“message=” +str);

ss.close();

} catch(Exception e) {

System.out.println(e);

}

}

}

//writing of a CLIENT program

Import java.io.BufferdReader;

Import java.io.BufferedWrite;

Import java.io.IOException;

Import java.io.InputStreamReader;

Import java.io.OutputStreamWriter;

Import java.io.PrintWriter;

Import java.net.ServerSocket;

Import java.net.UnkknownException;

Public class SocketClient {

Private String hostname;

Private int port;

Socket socketClient;

Public void connect() throws UnknownHostException, IOException{

System.out.println(“Attempting to connect to” +hostname +:” +port);

System.out.println(“connection is established”);

}

Public void readResponse() throws IOException{

String userInput;

BufferedReader stdIn=new BufferedReader(new InputStreamReader(SocketClient.getInputStream()));

System.out.println(“Response from server”);

While((userInput=stdIn.readLine())!=null) {

System.out.println(userInput);

}

}

Public static void main(String arg[]){

SocketClient client = new SocketClient(“localhost”,9990);

Try {

Client.Connect();

Client,readResonse();

} catch(UnknownHostException e) {

System.err.println(“cannot establishconnection.Server may not be up” +e.getMessage());

}

}

}

//Writing SERVER

Import java.io.BufferdReader;

Import java.io.BufferedWrite;

Import java.io.IOException;

Import java.io.InputStreamReader;

Import java.io.OutputStreamWriter;

Import java.io.PrintWriter;

Import java.net.ServerSocket;

Import java.net.Socket;

Public class javaSocket1 {

Private ServerSocket serverSocket;

Private int port;

Public javaSocket1(int port) {

This.port=port;

}

Public void start() throws IOException{

System.out.println(“starting the socket at port:” +port);

serverSocket=new ServerSocket(port);

//listen for clients block till one connects

System.out.println(“waiting for client”);

Socket client = serverSocket.accept();

//client has connected to th send welcome msg

Send welcomeMessage(Client);

}

Private void sendWelcomeMessage(Socket client) throws IOException

{

BufferedWriter writer=new BufferedWriterr(new OutputStreamWriter(client.getOutputStream()));

Writer.write(“hello. You are connected to a Simple SocketServer.what is your name?”);

Writer.flush();

Writer.close();

}

Public static void main(String[] args){

Int PortNumber=9990;

Try{

Javasocket1 socketServer=new JavaSocket(PortNumber)

socketServer.start();// creates a new object

}

Catch(IOException e) {

e.printStackTrace();

}

}

}

Some useful Servers:

\*this shows several servers you can build with server sockets.

The three different HTTP servers are presented each with a different special purpose

HTTP servers:

* It is a large protocol
* It must respond to requests for files, convert URL’s into filename
* Respond to post and get requests handler requests for file that does not exist

A Single Fileserver:

* HTTP server begins with a server that always sends out the same file, no matter what the request is it is called single fileHttpServer.

Redirector:

* It is a useful application for a special purpose HTTP server.
* It is called as JHTTP it can serve the entire document tree including, images, Applets, HTML files, Text files and more
* It is similar to single FileHTTP server
* These servers are light weight.

// Program for HTTP server

Package com.Stackoverflow.q3732109

Import java.io.IOException;

Import java.io.OutputStream;

Import com.sun.net.httpserver.HttpExchange;

Import com.sun.net.httpserver.HttpHandler;

Import com.sun.net.httpserver.HttpServer;

Public class test {

Public static void main(String[ ] args) throws Exception

{

HttpServer server=HttpServer.Create(new InetSocketAddress(8000),0);

Server.createContext(“/test”,new MyHandler());

Server.setExecutor(null);//creates a default executer

Server.Start();

}

Static class Myhandler implements HttpHandler{

@override

Public void handle(HttpExchange t) throws IOException

{

String response=”this is the response”;

t.sendResponseHeader(200,response.length());

OutputStream os=t.getResponseBody();

Os.Write(response.getBytes());

Os.Close();

}

}

}

To execute:

<http://localhost:8000/test>

Steps to access password protected sites:

1. Javas URL class can access sites that use HTTP authentication although you need to tell what username and password to use
2. Java does not provide support for sites that use non standard cookie-based authentication

Support can be provided by yourself using the url connection class to read and write the http headers where cookies are set and returned.

a