EXPLORING SECURE SOCKET CONNECTIVITY FOR
PERSONAL DIGITAL ASSISTANT

by

Radha Mahakali

has been approved

December 2002

APPROVED:

Dr. Timothy E. Lindquist

Dr. Harry E. Koehnemann

Richard O. Whitehouse

Supervisory Committee

ACCEPTED:

Dr. Timothy. E. Lindquist

Department Chair
ABSTRACT

In the current era of handheld wireless devices, people prefer doing things from wherever they are. There are many applications already available, but most of them are general-purpose applications like address book, calendar and so on. Different groups of people have different requirements. So, along with these general-purpose applications some special-purpose applications are also being developed. The application developed in this project is targeted at a specific group – Network Administrators. This application resides on a PDA (Personal Digital Assistant) and helps network administrators monitor a server’s performance from wherever they are. It also explores the use of secure communication in application development in Java.

It is a client-server application in which the client is a PDA and the server can be any server whose information is needed. Because network administrators use this application, there is a possibility of sensitive information being passed back and forth. Keeping this in mind, connectivity was made secure. Connectivity between the client and the server is done through secure sockets. There is a liaison program that sits on the server and is used to accept commands from the client, execute them on the server and send the results back to the client. The client is developed using J2ME™ (Java 2 Micro Edition) and the Liaison program is developed using J2SE (Java 2 Standard Edition). To obtain secure connectivity, SSL (Secure Socket Layer) protocol is used on the server side and KSSL (Kilobyte Secure Socket Layer) protocol is used on the client side.
TABLE OF CONTENTS

LIST OF FIGURES..................................................................................................................v

INTRODUCTION ....................................................................................................................1
- Background .........................................................................................................................1
- Problem .............................................................................................................................1
- Scope .................................................................................................................................1
- Approach .........................................................................................................................2
- Sequence of Presentation ..............................................................................................2

BACKGROUND LITERATURE .............................................................................................4
- Introduction to J2ME™ .......................................................................................................4
- Introduction to CLDC ......................................................................................................4
- Introduction to MIDP .......................................................................................................5
- MIDlet’s and MIDlet lifecycle .......................................................................................5
- Compiling and Preverifying a MIDlet .............................................................................6
- The MIDlet Suite ............................................................................................................7
- Application Manager ....................................................................................................7
- Manifest and JAD files ..................................................................................................8
- PRC Files ......................................................................................................................8
- About Wireless toolkit ..................................................................................................9
- About security protocols being used ............................................................................9
- About SSL protocol .......................................................................................................9
- About KSSL protocol ..................................................................................................10

METHODOLOGY .................................................................................................................13
- Background .....................................................................................................................13
- Need for this application ...............................................................................................13
- Setting up the Environment .........................................................................................14
- Architecture ..................................................................................................................14
- Class Diagram ..............................................................................................................15
- Client Implementation .................................................................................................15
- Server Implementation ...............................................................................................16
- Level of security provided .........................................................................................16
- Setting up the Emulator ..............................................................................................17
- Testing ...........................................................................................................................18

RESULTS ..........................................................................................................................26

CONCLUSIONS .................................................................................................................28
- Problems encountered and their solutions ................................................................28
- Conclusions ....................................................................................................................29

REFERENCES ....................................................................................................................30
LIST OF FIGURES

FIGURE 1: SSL PROTOCOL ..............................................................10
FIGURE 2: KSSL API .................................................................11
FIGURE 3: APPLICATION ARCHITECTURE ..................................14
FIGURE 4: SETTING UP THE EMULATOR ..................................18
FIGURE 5: SETTING UP THE ENVIRONMENT VARIABLES ..........19
FIGURE 6: COMPILING AND PREVERIFYING .........................19
FIGURE 7: CREATING THE JAR FILE ...........................................20
FIGURE 8: INVOKING THE CONSTRUCTOR ...............................20
FIGURE 10: STARTING THE SERVER .........................................22
FIGURE 11: STARTUP SCREEN ON THE CLIENT .......................23
FIGURE 12: COMMAND OPTIONS AVAILABLE ON CLIENT .......23
FIGURE 13: RESULTS DISPLAYED ON THE CLIENT .................24
CHAPTER 1
INTRODUCTION

Background

In this age of wireless communication, people prefer doing everything from their handheld devices like cell phones and PDA’s (Personal Digital Assistant). Many applications have been developed and are being developed for these devices to perform different tasks. These programs are for general purposes such as calendar, address book email programs and so on. There are different groups of people using these handheld and their needs may vary. Apart from these general-purpose applications, some special purpose applications can be and are being developed depending on necessity and demand. Considering a specific group like system/network administrators, it would be helpful if they could monitor the servers directly from their handheld without physically going to the server. The availability of necessary information on the handheld devices makes it easy for the administrator to have an idea of what is going on even if she/he is on the run. In this process, important information is sent back and forth between the handheld and the server, and it is essential that this information be passed securely.

Problem

Create an application for a PDA, which helps system administrators to monitor different servers securely.

Scope

This application is designed only for a specific group of people and is not a general-purpose application like an address book. What the application does is to contact
a server, extract some performance related information from the server and displays it on the PDA.

**Approach**

Java is becoming the most popular object oriented language. Also, network programming is made very easy in Java. These reasons made Java an obvious choice for the development of this application. The application is basically a client-server application in which the client is a PDA and the server is any server that needs to be monitored. Communication between them is done via secure sockets. The application has two parts, a program written in J2SE (Java 2 Standard Edition) [Java2™ Platform Standard Edition Documentation] that resides on the server that is being monitored and a program that is available on the client that is developed using J2ME™ (Java 2 Micro Edition) [Java2™ Platform Micro Edition]. The program that resides on the server acts as a liaison between the client and the server. It takes the commands sent by the client, executes them on the server and sends the results back to the client. This program uses the SSL (Secure Socket Layer) protocol [“Secure Networking In Java”] to make a secure socket connection. The client side program makes use of KSSL (Kilobyte Secure Socket Layer) protocol [“Securing the Wireless Internet Using “Kilobyte” SSL”], [“Securing J2ME[tm] Applications”] to make secure socket connections.

**Sequence of Presentation**

- Chapter 1, which is the introduction, gives a brief description of what the project is, what is its scope, some background about the topic and importance of the problem. It also gives the approach used to solve the problem.
• Chapter 2 gives a detailed description of the current knowledge available in this area and how it is being used in this application.

• Chapter 3 gives the methodology applied in implementing this application.

• Chapter 4 gives the results of this project.

• Chapter 5 gives the conclusions and any recommendations.
CHAPTER 2
BACKGROUND LITERATURE

Introduction to J2ME™

Recognizing that “one size does not fit all”, Sun Microsystems Inc. has regrouped its innovative Java™ technologies into three editions: Micro (J2ME™ technology), Standard (J2SE™ technology), and Enterprise (J2EE™ technology). Each edition is a rich set of tools and supplies that can be used with a particular product. [Java™ Platform Micro Edition]

J2ME™ (Java 2 Micro Edition) technology specifically addresses the vast consumer space, which covers the range of extremely tiny commodities such as smart cards or pagers. With the delivery of the J2ME™ platform, Sun provides a complete, end-to-end solution for creating state-of-the-art networked products and applications for the consumer and embedded market. J2ME™ technology enables device manufacturers, service providers, and content creators to gain a competitive advantage and capitalize on new revenue streams by rapidly and cost-effectively developing and deploying compelling new applications and services to their customers worldwide. [Java™ Platform Micro Edition]

Introduction to CLDC

The Connected Limited Device Configuration (CLDC) is one of the two configurations defined through the Java Community Process™ to be part of the Java™ Platform Micro Edition. CLDC is the foundation of the Java runtime environment that targets small, resource-constrained devices, such as mobile phones,
personal digital assistants, and small retail payment terminals. At the heart of CLDC is Sun’s K Virtual Machine (KVM), a virtual machine designed from the ground up with the constraints of inexpensive mobile devices in mind. Named to reflect that its size is measured in tens of kilobytes, CLDC is suitable for devices with 16/32-bit RISC/CISC microprocessors/controllers, and with as little as 160KB of total memory available-128KB of which is for the storage of the actual virtual machine and libraries themselves. Combined with a set of specific Java API (Application Program Interface), such as the Mobile Information Device Profile (MIDP), CLDC provides a complete J2ME runtime for small resource-constrained devices. [Java2™ Platform Micro Edition, Sun Community Source Licensing, Connected Limited Device Configuration].

Introduction to MIDP

The Mobile Information Device Profile is a set of Java2™ APIs, which, together with the CLDC provides a complete J2ME™ application runtime environment targeted at mobile information devices, such as cellular phones and two-way pagers. The MIDP specification addresses such as user interface, persistence storage, networking, and application model. The MID Profile provides a standard runtime environment that allows new applications and services to be dynamically deployed on the end user devices. [Mobile Information Device Profile, Sun Community Source Licensing].

MIDlet’s and MIDlet lifecycle

A MIDlet is the heart of a MIDP application and allows the device to start, pause and destroy the application. MIDlet is an abstract class that is sub classed to
define an interface between our application and the application management software
on the device. [“A simple MIDP Application”].

When a device receives a message to start an application, the application
management service on the device calls startApp() method. This does any
initialization that is required. But there is one problem associated with initializing
here, this method maybe called a number of times during the life of a MIDlet. For
example a MIDlet may be in paused state and to restart it startApp() is called. A
better place for initializations would be the constructor. A user might want to switch
between applications, and then one task has to be suspended to do the other. In such a
situation, the device calls the pauseApp() method. When the user tries to close an
application, the destroyApp() method is called. This method is invoked to allow our
application to clean up any resources it might be using like network or database
connections. Start, pause and destroy are the three stages in a MIDlet’s life cycle. [“
A simple MIDP Application”].

Compiling and Preverifying a MIDlet

Compilation is done using the standard javac compiler command. However,
since we are compiling the application for J2ME environment rather than J2SE, the
–bootclasspath option must be used. This takes advantage of Java cross compilation
capability, which allows Java compiler to target the class files for an environment
other than standard Java. The –g:none option is used to prevent debug information
from being included in the class files. [“A Simple MIDP Application”].

For security reasons, the standard Java Runtime Environment verifies every
class file before loading it into memory. This is done to make sure that the class file is
valid and does not attempt to access memory outside of its boundaries or access disk. Since J2ME must cater to devices that are limited than a desktop, some of the J2ME machines handle it in a different way. Verification is not entirely done on the device. Instead, as a part of the deployment process each class file must be preverified using a preverify utility provided in the J2ME development environment. This utility creates new class files in a directory called \output off of the current directory. To change the output directory, -d option can be used as with other Java utilities. [“A Simple MIDP Application”].

The MIDlet Suite

Although it is not a requirement to place MIDlets in a JAR (Java Archive) file, it is the most common means of distributing a J2ME/MIDP application. The MIDlet Suite is the name given to the entire collection of all files and resources that may be required as part of a MIDlet. The MIDlet Suite consists of

- Java class files enclosed in a JAR file,
- Manifest file describing the contents of the JAR,
- Resources (images, etc) enclosed in a JAR file, and
- Java Application Descriptor file (JAD).

[Muchow. “MIDlet Packaging with J2ME”].

Application Manager

The application manager is a software program that is device dependent and is implemented by the manufacturer. It is responsible for installing, running and removing MIDlets from the device. [Muchow.”MIDlet Packaging with J2ME”].
Manifest and JAD files

A manifest file describes the contents of a JAR file. It is stored in the JAR file itself and has the name manifest.mf.

As with the manifest, JAD (Java Application Descriptor) file contains information about the JAR file. There are two reasons to create a JAD.

- To provide information to the application manager about the contents of a JAR file. With this information, decisions can be made as to whether or not a MIDlet is suitable for running on the device.
- To provide a means for parameters to be passed to a MIDlet without having to make changes to the JAR file.

There are certain attributes that must be defined in the manifest as well as the JAD files. Also some attributes have to be same in both the files. If attributes are not defined, application manager will not load the MIDlet. If attributes are not equal, application manager will not load the JAR file. [Muchow."MIDlet Packaging with J2ME"].

PRC Files

A PRC file is kind of archive, like a Zip file. It stands for “Palm OS Resource Collection”. PRC files are mostly used to hold Palm applications. MIDP comes with a converter tool that allows us to convert preverified class files into palm executable PRC format. [Combee. “Secret World of Palm OS: PRC File Frequently Questioned Answers”].
About Wireless toolkit

The J2ME™ Wireless Toolkit is a set of tools that provides developers with the emulation environment, documentation and examples needed to develop CLDC/MIDP compliant applications. Version 1.0.4 also supports HTTPS. [Java2™ Platform Micro Edition, Wireless Toolkit 1.0.4].

About security protocols being used

The other aspect of the project is to provide secure communication between the client and the server. The server side makes use of SSL (Secure Socket Layer) protocol while the client uses KSSL (Kilobyte Secure Socket Layer) protocol. The reason for using two different protocols on the client and server is, SSL is a very resource-intensive protocol and here the client which is a PDA (Personal Digital Assistant) has limited resources. KSSL is specifically developed to provide secure communications for devices with limited resources.

About SSL protocol

Secure Socket Layer is a widely implemented and accepted standard for making network communications more secure. One of the reasons it is nice is that it presents a shrink-wrapped solution to the problem of secure network communications. Behind the scenes, a lot of tricky cryptographic stuff is going on, but the user need not worry about it. Netscape originally developed SSL. SSL is standard equipment in web servers and browsers. For any payment on the internet using a credit card, chances are SSL is already being used; web pages whose URLs begin with https are transmitted to your browser using SSL. SSL is a staple of electronic commerce. SSL and Java seem like a natural match. After all, Java has
excellent cryptography and security infrastructure. Until the release of J2SE (Java 2Standard Edition) 1.4; APIs for using SSL are available as a separate download called Java Secure Socket Extension (JSSE). APIs for SSL have become a part of J2SE1.4. [Knudsen. “Secure Networking In Java”].

SSL is a protocol that runs on top of a normal TCP/IP connection. When a network connection is first opened, SSL goes through a process called handshaking. Basically the two ends of the network connection negotiate to decide how they will encrypt the rest of the conversation. They exchange information about what cryptographic algorithms they have. With a little luck, the two sides have a common set of algorithms that can be used to encrypt the rest of the communication. An SSL Socket performs Handshaking when data is first sent or received over the network connection. SSL also includes the capability to authenticate one or both sides of the conversation. [Knudsen. “Secure Networking In Java”].

**FIGURE 1: SSL PROTOCOL**

[“Introduction to SSL”]

SSL is a protocol that runs on top of a normal TCP/IP connection. When a network connection is first opened, SSL goes through a process called handshaking. Basically the two ends of the network connection negotiate to decide how they will encrypt the rest of the conversation. They exchange information about what cryptographic algorithms they have. With a little luck, the two sides have a common set of algorithms that can be used to encrypt the rest of the communication. An SSL Socket performs Handshaking when data is first sent or received over the network connection. SSL also includes the capability to authenticate one or both sides of the conversation. [Knudsen. “Secure Networking In Java”].

**About KSSL protocol**

“Kilobyte” SSL (KSSL) is a small-footprint, client-side-only implementation of SSL v3.0 for handheld and wireless devices. The process of developing
connectivity applications for these devices requires consideration of the unique characteristics of a wireless environment, such as weaker CPUs, network latency, low bandwidth, and intermittent connectivity. [King “Securing the Wireless Internet Using “Kilobyte” SSL”].

Popular thinking dictated that the widely used Secure Sockets Layer (SSL) protocol was too heavy-weight and memory-intensive for securing wireless devices. To fill the need for wireless security, the WAP (Wireless Application Protocol) forum developed the WTLS (Wireless Transport Layer Security) protocol. Unfortunately, WTLS requires a proxy in order to work with the large installed base of web servers secured with SSL protocol. Since the proxy has to decrypt and re-encrypt the packets to translate between SSL and WTLS, the proxy can see all the messages, which compromises privacy. Given the unlikelihood that millions of SSL-secured web servers would use WTLS, Vipul Gupta and Sumit Gupta, working for a Sun laboratories project exploring security for small devices, developed an implementation of SSL for wireless devices. [King. “Securing the Wireless Internet Using “Kilobyte” SSL”].

FIGURE 2: KSSL API

The J2ME™ MIDP package, enhanced with KSSL, provides an easily programmable, mobile device platform consisting of a set of APIs that can be used to write an application requiring secure network transactions using SSL. To develop SSL-secured applications, it is necessary to have this enhanced version of MIDP package and a hardware platform that offers an underlying reliable, bi-directional byte-stream-oriented network connection. The J2ME package’s KSSL feature set:

- Implements client-side SSL v3.0
- Supports RSA_RC4-128_MD5 and RSA_RC4_40-MD5 cipher suites (most commonly used and fast)
- Provides server authentication via RSA signatures as well as arbitrary certificate chain lengths. The KSSL feature set (or simply “KSSL”) does not support client-side authentication, which is rarely used
- Supports caching of server certificates and SSL session reuse
- Includes a ksecurity package that uses the Java Card[tm] API to add functions not included in the base J2ME Platform, such as basic cryptographic functions, random number generation, encryption, and hashing.

[King. “Securing the Wireless Internet Using “Kilobyte” SSL”].

To summarize, when we look at the information, it is evident that a lot of research is being done in the areas of programming for handheld devices and also in the area of network security. Information is available for developers to make use of and implement these technologies in their applications. Let us see how these technologies are being applied in this application.
CHAPTER 3

METHODOLOGY

Background

Handheld devices are becoming popular these days and people are trying to use them as much as possible and their functionality is increasing day by day. Because anybody can use these devices, most of the applications that are available for these devices are general-purpose. There are some special purpose applications intended to do specific jobs depending on user requirement, but mostly they are being developed on demand.

Need for this application

So, what is the need for the application developed? A network administrator may have to go out of his work place, which means she/he cannot physically look at the server to check its performance. In that case, this application will provide a snapshot of the server on the administrator’s PDA (Personal Digital Assistant).

There is a commercial product called Big Brother that can be downloaded from the web. This program monitors System and Network-delivered services for availability. The current network status is displayed in a color-coded web page in near-real time. The idea for this project came up after looking at that program. This application is the first step in designing something like Big Brother for a PDA.
Setting up the Environment

All the necessary software has been downloaded and installed: J2SE 1.4 (Java 2 Standard Edition) for developing the liaison program that resides on the server, J2ME™ (Java 2 Micro Edition) for developing the client application. J2ME™ is comprised of CLDC 1.0.3 (Connected Limited Device Configuration) and MIDP 1.0.3 (Mobile Information Device Profile). The runtime environment needed to run the application needs to support secure sockets. The one that is available at Sun Microsystems Inc. website at this point does not support this feature. So, after some negotiations with Sun Microsystems Inc., a beta version of the runtime environment was obtained. This developer registered as a developer at palm website and obtained the ROM images for the emulator that is used for testing.

Architecture

This is a client-server application. The client is a Palm OS based PDA and the server is a web server. The developer implemented a MIDlet that resides on the PDA and another Java program that resides on the web server. The client and the server communicate using secure sockets.
Client Implementation

The MIDlet provides the user with an interface wherein the user can type in the URL for the server. Once the connection is made, the user will be provided a list of performance related commands that can be sent to the server and the results will be sent back to the client and displayed. It uses KSSL (Kilobyte Secure Socket Layer) protocol to open a secure socket connection.

In MIDP the general way to make a network connection is to create a Connector object using the Connection.open() method. But in this particular application, an SSLStreamConnection object is used instead. There seems to be
greater demand for HTTPS rather than raw SSL Sockets so sslsocket feature was not productised. If HTTPS is used, internally it uses SSLStreamConnection but because this application uses raw SSL sockets, the developer had to use SSLStreamConnection. SSLStreamConnection API is not a Java standard so it is not publicly documented. The client also implements a Handshake Listener to deal with authentication of the server.

Because MIDP does not support serialization, all objects are converted into bytes and these byte streams are sent and received using DataOutputStream and DataInputStream respectively.

**Server Implementation**

The Java program that resides on the server opens an SSL Server Socket to make a secure socket connection with the client. It uses DataInputStream to receive the byte stream sent by the client. Uses the readUTF() method to convert the byte stream into string. Then the server uses a Runtime object to execute the commands read from the byte stream. The results are sent back to the client using the writeUTF() method.

In order to execute this program, the developer had to generate a key pair using the java key tool. Then start the program specifying the private key. It is provided as a static run time argument.

**Level of security provided**

As mentioned before, the communication between the client and the server is done via secure socket connection. This is obtained using SSL protocol on the server and KSSL protocol on the client. SSL protocol optionally allows each of the
communicating party to ensure the identity of the other party. This is called authentication. Once the parties are authenticated, SSL provides an encrypted connection for secure message transmission. [Java™ Secure Socket Extension (JSSE) Reference Guide for Java™ Standard Edition 1.4]. The current application provides for encryption but not authentication.

In order to achieve authentication, the HandshakeListener can be modified to return true only if the MD5 hash of the received certificate matches the known MD5 hash of the server’s certificate. Another way is to create a class implementing the KeyStore interface making sure this keystore knows about the server’s certificate and passing this as an argument to SSLStreamConnection.setTrustedKeyStore () before opening the SSL connection.

**Setting up the Emulator**

The emulator has to be set up for supporting networking. The steps to do the set up are as follows:

- Right click on the emulator, select the “settings” menu.
- Now select the “properties” menu. This will bring up a screen. The “Redirect NetLib calls to host TCP/IP” box has to be checked. This redirects the networking calls through the hosts TCP/IP, instead of trying to connect with a modem.
- Select “OK”.

[“Simple Sockets”].
Testing

After implementing the client and the server, this developer created a batch file to accomplish a few tasks to generate an executable file for the client. This batch file sets up all the necessary environment variables, compiles the client program, preverifies it and creates a Jar (Java Archive) file that includes all the necessary resource files and a manifest. Then the batch file calls the converter to generate a PRC (Palm OS Resource Collection) file. This is the executable file that can be installed on an emulator or the actual device for testing. In order to generate a PRC file, the converter also makes use of a JAD (Java Application Descriptor) provided by the developer.
FIGURE 5: SETTING UP THE ENVIRONMENT VARIABLES

FIGURE 6: COMPILING AND PREVERIFYING
FIGURE 7: CREATING THE JAR FILE

FIGURE 8: INVOKING THE CONSTRUCTOR
Once the PRC file is generated, it can be installed on the testing environment, either an emulator or an actual device. Along with this, two more PRC files are installed. They are MIDP.prc and Debug.prc. The purpose of MIDP.prc is to provide the Java runtime environment. Debug.prc helps us to log any debugging information. This is how the log is enabled:

- Launch the Debug application. Tap on the “Show” and exit the application by tapping on the “Home” icon. This has the effect of enabling the Debug sub menu under Java HQ preferences.

- Launch the Java Manager (Java HQ) application. Tap on “Preferences”. This will bring up a dialog box titled “Java Preferences”. Set “Networking” to “Enabled”.

- Set “Preferences” to “Debug”. Set “Save Output” to “Yes”. This enables logging of all messages printed via System.out.println and System.err.println.
• Set “Preferences” to “Security” and choose “Disabled” for “Check Java Application at startup”.

• Tap on “OK” to save these settings before exiting “Java HQ.”

On the server side, the liaison program is compiled using the Java compiler. Because this program uses secure sockets, in order to execute the program, a private key is required which is generated using the keytool provided by Sun Microsystems Inc. This private key is provided as a static runtime argument and the program is started on port 8081.

![Image of server starting process]

**FIGURE 10: STARTING THE SERVER**

The Client also opens a socket on port 8081. Once a connection is established between the client and the server, commands are sent to the server and results are displayed on the client.
FIGURE 11: STARTUP SCREEN ON THE CLIENT

FIGURE 12: COMMAND OPTIONS AVAILABLE ON CLIENT
Throughout the development, an emulator was used to test the application. Once this developer was satisfied with the test results, the application was installed on a physical device. This developer obtained account on one of the committee members’ server to install the liaison program. A Xircom wireless card was used on the PDA and a wireless connection to the server was made and the application was tested.

In order to test this application, the liaison program has to be running on the server opening a server socket on port 8081.

Test was conducted to compare the time between a regular socket connection and a secure socket connection in terms of how long it takes for the connection to be established. Also test was compared to see the time difference to execute a command hundred times on a secure socket connection as well as a regular socket connection.
The reason for using J2SE1.4 is, that is the latest version released when this developer started developing this program, and also it has inbuilt APIs (Application Programming Interface) for creating secure socket connection on the server side using SSL (Secure Socket Layer) protocol. Same way CLDC1.0.3 and MIDP1.0.3 are the latest versions released when this developer started developing this application and they have inbuilt classes to create secure socket connection on the client side using KSSL (Kilobyte Secure Socket layer) protocol.

Developing this application gave this developer a chance to work with the latest technologies. Because the technologies are new, there were not very many examples available for reference, a few problems occurred in getting the environment set. But the problems were successfully solved and encountering these problems helped this developer to gain knowledge about the possible problems and how to go about solving them.
CHAPTER 4

RESULTS

The current project is an application for a Palm OS based Personal Digital Assistant that helps an administrator to monitor a server’s performance from anywhere if she/he has a wireless card to plug into the PDA to make a connection. Keeping in view the sensitivity of data that might flow; the connection has been made secure.

Also test has been done to do some time profiling. Time taken to make a regular socket connection is compared with the time taken to make a secure socket connection. Similarly, time taken to execute a certain number of commands with regular sockets is compared with the time taken for the same number of commands to execute on a secure socket connection.

The results of the current project are:

• Successfully made a secure socket connection to a server and send commands to the server and get the results back on the client- emulator as well as the actual device.

• Used a wireless card on the actual device and made a wireless connection to the server and got the application running.

• Observed that it takes 2257 milliseconds more time to make a secure socket connection compared to a regular socket connection on an emulator.
• Observed that it takes 9870 milliseconds more to execute a set of 100 commands on a secure socket connection compared to a regular socket connection on an emulator.

• Learned about the issues involved in using the beta version of software.
CHAPTER 5

CONCLUSIONS

With hacker’s world over finding increasingly ingenious ways to break into networks - wired as well as wireless, the necessity to develop secure applications has also increased. Threats ranging from denial of service attacks to compromise of important information are leading the application development community to consider new and often more expensive defenses. People realize the importance of security and a lot of research is being done to provide more reliable applications. The current project is developed with the intent of designing a secure application to facilitate people to get a snapshot of their server on a PDA (Personal Digital Assistant). The application explores a security protocol called KSSL (Kilobyte Secure Socket Layer) developed at Sun Microsystems Inc. research lab for J2ME and compares the performance (in terms of time) of a secure socket connection to a regular socket connection on small and resource-constrained devices.

Problems encountered and their solutions

- Sun Microsystems Inc. has changed the class file format in J2SE1.4, which is not compatible with the class files generated by J2ME™. This resulted in a constant index pool error during preverification. This was resolved by using one of the options that comes with the Java compiler (javac). The option is \(-\text{target}\), which lets users specify the class file format they want to use.

- The newer version of the PRC (Palm OS Resource Collection) Converter tool provided with MIDP1.0.3 mandates the user to provide the exact size (without
variation of even a single byte) of the JAR (Java Archive) file in the JAD (Java Application Descriptor) file. This was not the case with earlier versions of the converter. Until this was figured out, PRC file could not be created.

- Java runtime environment for the PDA that is available at Sun Microsystems inc. does not support raw secure sockets. So, a beta version of the runtime environment has to be obtained from Sun Microsystems Inc.

- The biggest problem was to get the secure socket connectivity to work. The open() method of the Connector class was better suited to implement HTTPS rather than raw sockets. So, the SSLStreamSocket has to be used instead. This was done with the guidance of Vipul Gupta of Sun Microsystems Inc. who is one of the designers of KSSL.

Conclusions

Based on the experience working on this project, the following conclusions can be reached.

- Handheld devices are the way of future and J2ME (Java2\textsuperscript{TM} Micro edition) has the potential to cater to applications for those devices.

- KSSL is a very useful protocol for limited device applications, but lack of documentation (which is not uncommon for a product still under development) is making it difficult to program its use.

- There is a performance penalty to be paid for using KSSL. The performance loss is reasonable for problems requiring secure communication.
REFERENCES


   http://bb4.com/


http://java.sun.com/products/j2mewtoolkit/


http://java.sun.com/j2se/1.4/docs/guide/security/jsse/JSSERefGuide.html


http://dcb.sun.com/practices/howtos/kssl.jsp


http://wireless.java.sun.com/midp/articles/security/
17. “Mobile Information Device Profile, Sun Community Source Licensing”.
   Retrieved from the World Wide Web:
   
   http://www.sun.com/software/communitysource/midp/

   

   
   http://webdev.apl.jhu.edu/~rbe/kvm/tutorial/SockTest.html