FRAMEWORKS FOR SECURING LIMITED-DEVICE APPLICATIONS

Timothy Lindquist *Aarthi Ramamurthy* *Ramon Anguamea*
Timothy.Lindquist@asu.edu *Aarthi.Ramamurthy@asu.edu* *Ramon.Anguamea@asu.edu*
Division of Computing Studies
Arizona State University

Abstract

In this paper, we compare the features available for developing secure distributed applications for limited devices, such as smart phones. We limit our scope to examine frameworks for Java. This work is part of a continuing project which is considering capabilities and performance for application development on these platforms. The paper considers performance as it relates to various approaches to securing applications.

The paper addresses two separate concerns. First is protecting access to resources by an executing application. The facilities for defining, limiting and controlling applications during their development, installation and execution are described. Second, we discuss approaches available for securing communication among application components running on servers or limited devices.

1. Introduction

In this paper we consider limited devices that are connected to the Internet and other communication media, for example, handheld devices such as intelligent cell phones and PDAs with Internet connections or platforms which combine these functionalities. These devices have limited memory, limited processing power, no hard disk storage, small display screens, and limited human input capability. We consider only those having communication facilities (WiFi or EV-DO)..

The operating environments for these devices are comprised of three base components: local operating system, network operating system and language runtime environment. The leading operating systems for these devices are Symbian, Palm and Windows Mobile 6. Connected limited-device configuration and mobile information device profile (CLDC1.1/MIDP2) are the Java frameworks designed for resource constrained devices, such as phones and PDA's. CLDC1.1/MIDP2 as realized by the IBM J9 runtime and SUN Wireless toolkit pre-defined classes, is the security environment we evaluated for this paper. Various development environments are available depending upon platform and language. For Microsoft Windows Mobile 6 the application development environment for the .NET languages, such as C#, is the .NET Framework together with Visual Studio 2005 with the Compact Framework 2.0. Several alternatives are available for configurations utilizing Java, in part depending on the Java runtime environment being used. Sun's CLDC HotSpot and IBM's J9 are two popular Java runtime environment choices. Add-on packages and various configurations are available to support different security approaches, device capabilities and networking needs.

2. Background

The connectivity of computing devices to the Internet, has enabled malicious attacks. The motivation for attacks varies from willful espionage to experimentation. Equally important to protection from attack is the ability to prevent harm from mistakes in coding, configuration or user operations. Protection and detection are difficult in handheld devices because of limited capability.

Trust is confidence in expected functionality. When running an application the user must trust that it produces valid information and that privacy, integrity, or confidentiality will not be compromised. There are several security policies, protocols and mechanisms that are of particular interest to the limited devices. Languages such as Java, C#, and other scripting languages are widely used in distributed applications and provide varying degrees security support. Java and C# both permit examination of compiled intermediate code for unsafe actions. Both the Java CLDC/MIDP and Mobile 6 execution environments support an array of cryptographic functions that can be used

1530-1605/08 $25.00 © 2008 IEEE

by communication protocols and by the system elements to aid in access control at the device and application level.

3. Java CLDC/MIDP

Three different limited device configurations exist for the Java 2 Micro Edition (J2ME). For more capable devices such as set-top boxes and high-end wireless devices the Connected Device Configuration (CDC) defines an API whose functionality is close to J2SE, but is reduced as appropriate for the limited hardware and applications. At the lowest level of functionality is the JavaCard API (for Smart-cards/Sim-cards). JavaCard as can include functionality for asynchronous security operations, such as encryption, decryption, digital signature, verification and others for limited devices whose computing capacity is unable to perform such operations without disrupting user-functionality. The Connected Limited Device Configuration (CLDC) is defined for PDA and wireless phone devices. A device such as a PDA or smart-phone running Java applications would include a virtual software stack with the following components:

- Mobile Information Device Profile (MIDP2) that supports the application life-time model, persistent storage, network resources and the user-interface.
- CLDC1.1 that supports the core Java language, IO and networking classes, security features and internationalization facilities.
- The selected Java runtime environment
- The device operating system and related services

3.1 Application Security Model

The J2SE model for securing the operations in an executing virtual machine changed dramatically as Java evolved. Java originally, used the sandbox model for application security. Initial versions of Java provided full trust to classes loaded locally and prohibited all sensitive operations from any code obtained dynamically.

Java1.2 introduced support for a continuum of access control. Access to system resources (such as files, sockets, runtime, properties, security permissions, serializable, reflection, and window toolkit) is granted based on domains. A domain is defined to include:

- a set of permissions (resources),
- a set of operations associated with each permission,
- codebase indicating the code origin,
- a digital signature of the code which allows identification of the signer and verification that the code has not been modified.

The codebase indicates the file or URL from which the code is loaded. If signed, the alias of the public key can also be used to define a domain. Each class loaded into a Java virtual machine has an associated protection domain, which defines the access it has to resources. When execution encounters an operation that requires a system resource, all classes representing the currently executing methods (contents of the runtime stack) are checked to assure all have access to the resource. The Figure below is taken from the On-line Java Tutorial and shows how an execution can include a range of protection domains ranging from no access to resources to full access.

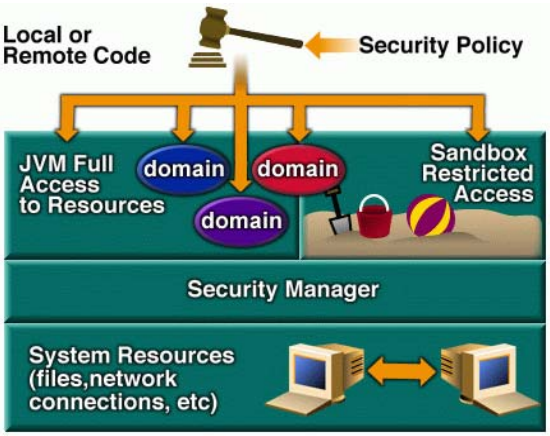

Figure 1. Controlling Access to Java Resources

In J2SE (Java2), security domains are defined by a policy file granting permissions to the domain. For example, suppose the company **GrowthStocksExpress** publishes an applet on their (hypothetical) web site at the URL:

http://GSE.com/applets

Assuming the applet needs connections to one or more hosts having a domain address ending with **GSE.com** on ports beginning at 2575, a policy for clients who access the applet may be:

```
grant signedBy "GrowthStockExpress", codeBase
"http://GSE.com/applets"
{     permission java.net.SocketPermission
```

"*.GSE.com:2575-", "accept, connect, listen, resolve";
};

A policy may consist of one or more **grants** each defining different domains. Each domain may have one or more associated permissions.

CLDC/MIDP2 security. The application security model for CLDC/MIDP2 draws on the model for J2SE, in that it includes domains and signed code. CLDC/MIDP2, however has a simpler model, in part because of the constraints imposed by the configuration and profile. The following CLDC/MIDP2 constraints are most significant

- Java Native Interface (JNI). JNI provides J2SE applications access to native code running on the platform. CLDC provides similar capabilities in Kilo Native Interface (KNI), but prohibits dynamically loading and calling arbitrary native functions.
- No reflection, remote method invocation or serialization. In J2SE, an RMI server or client can cause remote code to be automatically downloaded and executed to satisfy argument or return (sub)classes. When a serializable RMI parameter is provided an argument of an extended type, the RMI system will attempt to load (if necessary from an http codebase) the needed class.
- No user-defined class loaders. Related to the constraint above, the developer cannot define a class loader in CLDC. The class-loader in CLDC cannot be extended or replaced by the developer. A CLDC/MIDP2 application can only load classes from its own (signed) Java archive. As a result, the developer cannot extend or modify any classes in the CLDC configuration, MIDP2 profile, or which are provided by the runtime environment vendor.
- CLDC supports multi-threading, but it does not provide facilities to build daemons or thread-groups.

MIDP2 security protects access to sensitive API's by permissions. Protecting resources includes the concept of a domain, which is conceptually similar to J2SE. The full scope of protection includes the following elements:

- Protection domains (4) that are statically defined in a policy file (by the vendor) and associated to resource permissions; for example, socket, http, https, PushRegistry. The protection domains are Minimum, Maximum (or Trusted) and Untrusted.
- Certificate and archive signature. The jar file containing application class files and other resources can be digitally signed.
- Level of access – either Allowed or User.

Unlike J2SE, the 4 protection domains are device-specific and defined by the runtime vendor. They can be modified only as provided by the vendor. Each of the four domains is associated with a set of permissions together with a level of access. The 4 protection domains are defined by the runtime vendor.

- Minimum. None of the permissions are allowed.
- Maximum. All of the permissions are allowed.
- Trusted. All of the permissions are allowed.
- Untrusted. To be allowed, the user must provide consent.

The permissions defined by the MIDP2 specification include: http, socket, https, ssl, datagram, serversocket, datagramreceiver, and PushRegistry (invoke other applications). These permissions may be grouped together by the vendor into meaningful subsets and assigned to domains based on the subsets; for example, NetworkAccess.

Within a domain, the level of access may be different for different permission (sets). The accesses are:

- Allowed. The permission (set) is allowed without involving the device user.
- User level. The application's access to the permission(set) depends on explicit authorization from the device user.

With user level of access, a dialog box is presented to the user indicating information about the permission and asking the user

whether access should be granted. User level access can be specified on one of 3 modes:

- Oneshot. The user must be prompted for each operation on the protected resource.
- Session. When the user grants access, it applies to all operations on the resource during a single execution of the MIDlet.
- Blanket. When the user grants access, it applies to all operations on the resource during any execution of the MIDlet.

CLDC/MIDP2 provides MIDlet access to the Record Management System (RMS), which provides persistent storage for application data via a record store. MIDP2 provides shared access to the record store of other MIDlet suites, and provides that access should be provided as read-write or read-only.

Low-level security is provided by the J2ME Java virtual machine. A virtual machine supporting CLDC must reject invalid class files. This is accomplished by a two-step process. At development time, classes are pre-verified by a tool which adds special attributes to class files to facilitate runtime class verification on the device. Much of the verification process can be handled statically by the pre-verifier. At runtime, the virtual machine rejects classes that have not been pre-verified.

3.2 Security and Trust API

MIDP2 provides HTTPS and SSL for secure communications with other devices. But, runtime environment providers are increasingly providing additional options. For example, IBM's J9 version 5.7 provides web service security package which allows web method calls using encrypted SOAP envelopes or digitally signed method calls for authentication and information integrity.

Security and Trust Services API (SATSA) provides access to more comprehensive hash code, digital signature/verification, key/certificate management, as well as encryption and decryption. SATSA is designed as 4 optional components. The primary purpose is to provide access to a SmartCard Java device, which provides security functionality in an asynchronous manner that does not disrupt applications supporting the device user.

SmartCard includes the Java Card Protection Profile. The protection profile supports both open and closed cards. Open cards provide the end-user with the ability to install or activate new applications on the card. Closed cards have applications set by the vendor at the time the card is personalized for the end-user. A good example of a closed card may be a banking card that supports personal electronic purchases and bank account functions. Open cards that allow new applications to be downloaded and installed on the card present special security risk that would exclude open cards that include banking applications. Nevertheless, applications for open cards that support other aspects of security may become increasingly important. An example may be securely communicating information outside of direct e-commerce applications. Data integrity and authentication are becoming increasingly important as electronic communication proliferates.

The Java Card Protection Profile defines four different configurations for a Java Card based on open and closed cards. The minimum configuration corresponds to a closed card in which no applications can be installed on the card after it's been issued to an end-user. The three remaining configurations provide additional functionality that's available through the evolution of the Java Card specification, such as RMI (a limited version), logical channels, applet deletion, object deletion, external memory, biometry, and contactless interface. The Java virtual machine for the device includes an API (RTE API) that may contain classes for performing security operations on information and for certificate and key management.

SATSA runs on the limited-device, not on the Java Card. SATSA provides an interface to card security functionality, or when there is no associated smart card, provides security operations for the limited-device. SATSA has four optional packages.

- SATSA-APDU provides low-level stream/socket-based protocol for communicating between the limited-device and the card.
- SATSA-JCRMI provides an RMI interface that allows an application running on the limited-device to call methods running in applications on the Card. This interface would be used to instead of SATSA-APDU to avoid the overhead of programming with a low-level socket data protocol.
- SATSA-PKI allows limited device applications to use the smart card to digitally sign information or to verify digital signatures.

PKI also provides for key and certificate management.

- SATSA-CRYPTO. When a Java Card is not available, the CRYPTO API is used to compute security operations directly on the limited-device.

Use of APDU, JCRMI, and/or PKI is accomplished using threading on the limited-device. Threading allows security operations to take place on the card while other applications continue to run on the device supporting the end-user. In this scenario, SATSA is appropriate for limited-devices with constrained processing power. Independent of processing capability, using a Java Card may be necessary to provide assurance level that is appropriate to the application. The open-device nature of cell phones and PDA's make it difficult to certify trustworthiness of applications on the device. Instead, we can isolate all high-risk user-specific information and computations to a certified secure Java Card.

4. Secure Web Services

Web services provide an XML-based service protocol for communicating among components of a distributed application. Web services differ from prior similar technologies, such as Microsoft DCOM, Object Management Group CORBA and Java Remote Method Invocation through reliance on http protocol and XML.

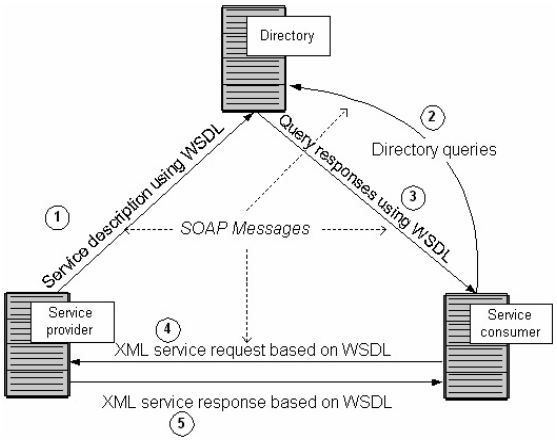

Figure 2. Web Services Architecture [9]

In Figure 2, the *service*, is performed by a web server acting as a container for executing the service code. This is generally just a web like page that gets posted similar to the way other http web requests are done. Instead of returning a HTML page, it returns an XML message in Simple Object Application Protocol (SOAP) format.

The service description - specified in Web Services Description Language (WSDL) - this description defines the web methods (functions) that a service will accept - the inputs that go into these methods, and the format of the output that can be expected in return. This is used in generating a web service client proxy class for the limited device.

The web service registry - is a directory of web services. The directory is optional because a web service need not be listed in a registry to be used. The registry provides a catalogue of available services - similar to Java Naming and Directory Service (JNDI).

The web service client proxy – The proxy negotiates the communication between a limited device client and the web service. It marshals arguments, signs or encrypts as appropriate, posts the message and interprets the result.

4.1 Types of Security Services

The following are the security services that may be required by a distributed limited device application.

Authentication: Ensures that the sender and receiver are who they claim to be. Mechanisms such as username/password, smart cards, and Public Key Infrastructure (PKI) can be used to assure authentication.

Authorization or Access Control: Ensures that an authenticated entity can access only those services they are allowed to access. Access control lists are used to implement this.

Confidentiality: This assures that information in storage and in-transit are accessible only for reading by authorized parties. Encryption is used to assure message confidentiality.

Integrity: Ensures that information, either in storage or in-transit cannot be modified intentionally unintentionally. Digital signatures are used to assure message integrity.

Non-repudiation: Requires that neither the sender nor the receiver of a message be able to legitimately claim they didn't send/receive the message.

4.2 Transport Level Security

The most popular security scheme for web services is SSL (Secure Socket Layer), which is typically used with http, and is supported by

CLDC/MIDP2. However using SSL for securing web services has a number of limitations. The inadequacy of SSL can be easily explained by a simple example.

Consider a Web Service that can be provided indirectly to a user. A user accesses a website which indirectly invokes another remote web service. In this case, we have two security contexts:

1. Between the user and the website
2. Between the user and the web service

The second security context requires the security of SOAP request/reply message (between the web site and the web service) to be assured over more than one client-server connection. SSL is inadequate to provide this type of security mainly because of the fact that while it encrypts the data stream, it does not support end-to-end confidentiality.

The shortcomings of SSL (https) should be considered when being used for a distributed application to reside on a limited-device.

SSL is designed to provide point-to-point security. Often, Web services require end-to-end security, where multiple intermediary nodes could exist between the two endpoints. In a typical Web services environment XML-based business documents route through multiple intermediary nodes.

Https in its current form does not support non-repudiation well. Non-repudiation is critical for business Web services and, for that matter, any business transaction.

Finally, SSL does not provide element-wise signing and encryption. For example, if there is a large purchase order XML document, yet only a single element, say, a credit card element needs to be encrypted. Signing or encrypting a single element is difficult with transport level security.

4.3 XML Signature

XML based security schemes, provide unified and comprehensive security functionalities for Web Services. The important ones being, XML Signature, XML Encryption, WS-Security (Web ServicesSecurity).

The W3C (World Wide Web Consortium) and the IETF (Internet Engineering Task Force) jointly coordinated to generate the XML digital signature technology. The XML digital signature specification [10] defines XML syntax for representing digital signatures over any data type. It also specifies the procedures for computing and verifying such signatures.

Another important area that XML digital signature addresses is the canonicalization of XML documents. Canonicalization enables the generation of the identical message digest and thus identical digital signatures for XML documents that are syntactically equivalent but different in appearance due to, for example, a different number of white spaces present in the documents.

The advantages of using XML digital signature can be summarized as below.

- It accounts for and takes advantages of two existing and popular technologies, viz., the Internet and XML.
- XML digital signature provides a flexible means of signing. For example, individual item or multiple items of an XML document can be signed. This becomes extremely useful in a scenario where each person in a workflow is responsible ONLY for certain work.
- It supports diverse sets of Internet transaction models. For instance, the document signed can be local or even a remote object, as long as those objects can be referenced through a URI (Uniform Resource Identifier). A signature can be either enveloped or enveloping, which means the signature can be either embedded in a document being signed or reside outside the document.
- It provides important security features like authentication, data integrity (tamper-proofing), and non-repudiation.
- XML digital signature also allows multiple signing levels for the same content, thus allowing flexible signing semantics. For example, the same content can be semantically signed, cosigned, witnessed, and notarized by different people.

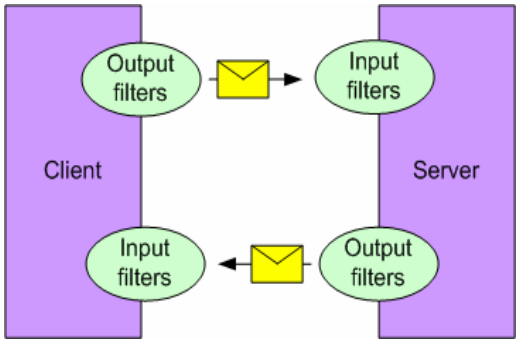

Figure 3. WSE – Input/Output filters [1]

Web Service Deployment Descriptor (WSDD) and Handlers play the pivotal role in the implementation of digital signature in Java. A deployment descriptor specifies aspects such as handlers and communication protocol. The Handler is a java class (implementing the Input and output filters of Figure 3) that provides a MessageContext through which access is provided to the input/output stream of XML \In the Apache AXIS framework, MessageContext is a structure, that contains: 1) a request message, 2) a response message, and 3) a number of properties.

All the SOAP message manipulation is done within the handler class.

6. Conclusions and Issues

The authors have continuing efforts in this area which include obtaining devices, software development environments, and simulators / emulators related to securing limited-devices. Our approach considers the operating environment on the device as well as their applications.

We are in the midst of consolidation of small hand-held devices to provide integrated functionality. Common applications including personal organizers, cell-phones, and multi-media players can effectively be placed on a single platform. While users who desire more than one of these functionalities are exploring integrated solutions, the industry is pushing separation (partly for financial reasons.) Consolidated functionality brings a higher diversity of applications onto limited devices, as does special purpose applications (for example autonomous vehicle control). Either way, security concerns increase.

The use of smart cards in the United States is just beginning after lagging behind use in some other regions. The integration of smart cards (SIM-Cards) on cell phones is an indication of this trend. Enabling high-risk applications, such as banking and purchasing, by leveraging smart cards integrated with other devices presents an attractive alternative. Of course the concern for security places new demands on platforms in which security has not historically been a high priority.

Performance has been the primary impediment to the use of more strongly object-oriented languages such as Java for limited device applications. Securing a distributed application complicates the issue. Important considerations include:

- Underlying architecture processor performance, ancillary processing capability such as SmartCard,
- Frameworks supporting securing the application, as well as communications,
- Use of security mechanisms appropriate to application needs (authentication, integrity, confidentiality),
- Proper use of available frameworks including proper handling of passwords, certificates, keys, digital signatures, and encrypted information.

Frameworks discussed in the paper are an important enabler to developing more secure distributed limited-device applications. Further usage reports and benchmarking for security mechanisms would better support developers.

References

[1.] Tim Ewald, "Programming with Web Services Enhancements 1.0 for Microsoft.NET", Available, see: http://msdn.microsoft.com/webservices/building/wse/default.aspx?pull=/library/en-us/dnwse/html/progwse.asp

[2.] Sun Microsystems Java Security and Crypto Implementation, http://www.cs.wustl.edu/~luther/Classes/Cs502/WHITE-PAPERS/jcsi.html

[3.] Knudsen, Jonathan; Understanding MIDP 2.0's Security Architecture. http://developers.sun.com/techtopics/mobility/midp/articles/permissions/

[4.] WebSphere Everyplace Micro Environment v5.7; MIDP Installation guide for J9 Palm runtime environment. Available online from IBM.

[5.] Mourad Debbabi, Mohamed Saleh, Chmseddine Talhi and Sami Zhioua: "Security Evaluation of J2ME CLDC Embedded Java Platform", in Journal of Object Technology, (5,2) Mar-Apr 2006. pp. 125-54.

[6.] Security and Trust Service APIs for Java Platform Micro Edition Developers Guide. Available from http://www.java.sun.com/

[7.] Pannu, K.; Lindquist, TE; Whitehouse, RO; and Li, YH; "Java Performance on Limited Devices"; Proc The 2005 International Conference on Embedded Systems and Applications, CSREA Press, Las Vegas, June, 2005.

[8.] Lindquist, TE, Diarra, M, and Millard, BR; "A Java Cryptography Service Provider Implementing One-Time Pad"; Proc. 37th Annual Hawaii Int'l Conf on Systems Sciences, ACM, IEEE Computer Society, January 2004.

[9.] Online Documentation on Web Services, Available from: http://www.service-architecture.com/web-services/articles/web_services_explained.html

[10.] XML Digital Signature Specification, W3C Recommendations, Available from: http://www.w3.org/TR/xmldsig-core