Security Considerations for Distributed Web-Based e-commerce Applications in Java

Timothy E. Lindquist
Electronics and Computer Engineering Technology
Arizona State University East
http://www.east.asu.edu/ctas/ecet
Tim@asu.edu

Abstract

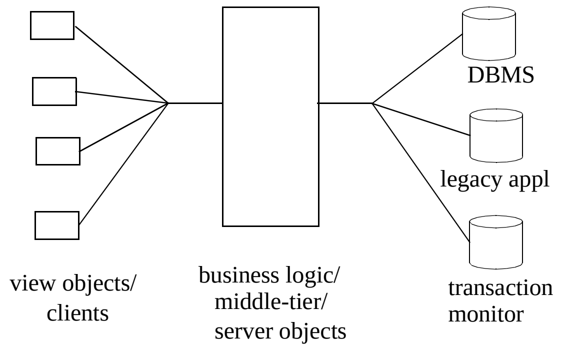
Today's distributed e-commerce applications typically rely upon various technologies in their realization, including the web, scripting languages, server-side processing and an underlying database. The combination of these technologies creates a system that requires attention to the security issues of each component and the system as a whole. In considering the overall system, issues arise from the interactions of security frameworks available for each component. In this paper, we consider the approach and related issues for distributed e-commerce applications developed with Java. The flexible nature of Java allows migration of objects (compiled code with state) through features such as RMI and Applets. Security for distributed applications developed in Java has issues and lessons applicable to systems of components built on different technologies.

1. Problem

Web-based e-commerce and distributed applications are changing the way we buy goods, access information and learn. Use of email and other related technology increasingly facilitates collaboration and is more commonly being used for official communications. Official communications via the internet are too often done in insecure mode.

Web-based e-commerce applications commonly employ multiple tiers (3-tier client server architecture) and a combination of technologies such as HTML, XML, JavaScript, Java (JSP, Servlets), ASP, dynamic html, CGI, and relational databases, as shown in Figure 1. Each of these technologies have separate and in some cases incompatible approaches to protection against intrusion.

For web-based applications, the communication between clients and the middle-tier is via web protocol http. Clients may employ any number of technologies such as applets, html, xml, and scripts. The middle-tier business



IGURE 1. 3-Tier Client-Server Architecture

logic often employs any of a number of CGI work-arounds such as Netscape's NSAPI, Microsoft's ISAPI, WebObjects, ASPs, Java J2EE, servlets and JSP. The combination of different technologies at each tier, presents special challenges to security of the overall application.

Development time and cost pressures often short-change security concerns. Problems range from software design constraints that prevent adequate security to insufficient testing to exercise common attacks. Often performance concerns limit the extent to which assurance can be implemented in a web-based application.

Emerging technologies and applications are also presenting new challenges to secure applications. Distributed Object technologies have been maturing for the past several years and are being increasingly utilized in web-based developments. Although some researchers have supported an approach where an object-web replaces the largely data-centered web of today, this has not materialized. Nonetheless, object technologies such as CORBA, DCOM and Java RMI enjoy increased usage in distributed web-based applications. Additional frameworks such as JINI, JavaSpaces, JNDI, and EJB support distributed Java Objects.

Distributed objects are network aware objects. Applications running on remote machines may communicate with each other at the object-level using this technology. With a remote reference to a Java RMI object, methods can be requested of the object (messages) in the same manner as if the object were local to the executing application. The SUN implementation of Java (sdk/jdk1.2 and up) includes a flexible and feature rich approach to security providing the basis for distributed Java applications.

2. Security Concerns

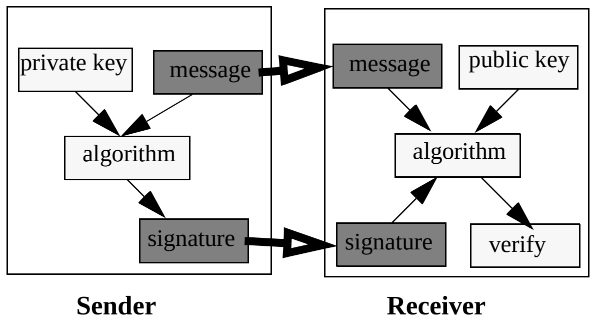
The platform independence of Java has lead to easy movement of (compiled) code across the internet. While the approaches of OMG CORBA (see: http://www.omg.org) and Microsoft do provide multi-language solutions, they do not provide the same code migration capabilities as is available with Java. Interacting remote Java objects may easily be written in a manner that requires dynamic movement and execution of code (class files) across the internet.

Security concerns include authentication, integrity and encryption/decryption. These may all come into play whenever information (code or data) is moved (across a network or within a single machine).

1. **Authenticity** allows the receiver of information to know with certainty the identity of the sender.
2. **Integrity** allows the receiver to know with certainty that information transmitted by the sender has not been modified or tampered with enroute.
3. **Encryption** is the process of taking data (called clear text) and a short key and producing cipher data that is meaningless to anyone who does not know the key. **Decryption** is the process of taking cipher data and a short key to produce the corresponding clear text.

Each of these basic security concerns come into play with distributed applications, for controlling executing applications as well as access to information. Figure 2 shows how authentication and integrity can be provided using digital signatures.

The sender uses his own private key (which must be kept protected utilizing access control) and together with a message to generate a digital signature, which is unique to the message and private key. The message and signature are transferred to the receiver. The receiver must have a public key (usually received separately) corresponding to the sender's private key. The public key can be used to verify a signature, but cannot be used to generate a signature. Upon receipt, the message and signature are verified assuring both authenticity and integrity of the exchanged message.



IGURE 2. 3-Tier Client-Server Architecture

3. Digital Signatures

Keys are generated in pairs. The private key is used to generate the signature and is kept confidential to whoever is doing the signing. The public key is used by the receiver to verify authenticity of the message. The signer should distribute the public key to anyone who will receive signed information.

The issue as to whether the public key actually corresponds to the sender is resolved with certificates. A certificate represents a chain of trust leading from the sender to the receiver, indicating that the public key belongs to whom you want to believe it belongs. If the sender and receiver both trust the same certificating agency then the chain may be of length one. Each link in the chain is a certifying agency (such as VeriSign or Entrust) which certifies that the entity prior to it in the chain (the owner of the private key or another certifying agency) is who they say they are.

Users should understand how certificates are signed and managed. Current web browsers display information about the certificate and who signed it, but few users ever look beyond the lock icon on their web browsers. This provides some opportunity for anyone with a signed certificate to use a man-in-the-middle attack. Simple possession of a certificate says nothing of integrity, quality or functionality of code or other information conveyed by the certificate holder.

Another complication of digital signatures is management of a certificate revocation list. Once a key is known to be compromised, there must be some way to inform users that it should no longer be trusted.

The SUN implementation of Java comes with a primitive set of tools for manipulating keys, certificates and digital signatures. It also includes the framework classes (in the package java.security) for program creation and verification of digital signatures.

Figure 3 includes a sample of Java that may exist for the sender. The example generates a public and private key pair

```
//generate the private and matching public key
KeyPairGenerator keyGen=KeyPairGenerator.getInstance("DSA",
                                                      "SUN");
SecureRandom random = SecureRandom.getInstance(
                                          "SHA1PRNG", "SUN");
keyGen.initialize(1024, random);
KeyPair pair = keyGen.generateKeyPair();
PrivateKey priv = pair.getPrivate();
PublicKey pub = pair.getPublic();
//create the signature object
Signature dsa = Signature.getInstance("SHA1withDSA", "SUN");
dsa.initSign(priv);
//read the datafile;
FileInputStream fis = new FileInputStream(args[0]);
BufferedInputStream bufin = new BufferedInputStream(fis);
byte[] buffer = new byte[1024];
int len;
while (bufin.available() != 0) {
        len = bufin.read(buffer);
        dsa.update(buffer, 0, len);
}
bufin.close();
//generate the signature
byte[] realSig = dsa.sign();
//save the signed data in a file
FileOutputStream sigfos = new FileOutputStream("sigOf"+args[0]);
sigfos.write(realSig);
sigfos.close();
```

FIGURE 3. Snipet of Java to sign a file

and uses the private key to generate a digital signature for a data file. The signature is saved to file. This code represents simplistically what must be done by the sender. The public key, the signature file and the data file are all transmitted to the receiver, where a similar program verifies the signature using the data and public key. The primary vulnerability of this approach rests in communicating the public key. An attack may replace the data, signature file and public key if they are all three transmitted together. Certificates are the most common mechanism used to assure the public key authentically identifies the sender. Good practice dictates that the public key be transmitted separately in an assureable manner. The public key (certificate) is stored by the receiver for later use to authenticate multiple subsequent transmissions.

This mechanism can be used to verify the authenticity and integrity of either data or program code that is transferred in distributed e-commerce applications. The approach verifies that information came from the purported sender and was not modified in transmission. Encryption is necessary to protect information from reading by others during transmission, as discussed below. Security within an executing Java application is based upon authenticity and integrity using digital signatures.

4. Securing Java Applications

Many aspects of Java's design lend well to distributed applications. One such example is serialization. The ability to externalize objects from one executing Java program (virtual machine) and to read them into another is a process Java calls serialization. Serializable objects may be transmitted through the internet without loss of object properties, including methods.

To accomplish object externalization, it is often necessary to move the compiled code along with object data. Several mechanisms exist within Java to do this either implicitly or explicitly under programmer control. Applets and Remote Method Invocation (RMI) are two such mechanisms. Applets are small Java programs communicated from a web-server and executed by a virtual machine running in the browser. RMI, provides the programmer with an object view of internet objects so that method calls, for example can formulated as though the object were in the same virtual machine. RMI capabilities are similar to Microsoft DCOM and the Object Management Group's Common Object Request Broker Architecture (CORBA).

Java's platform independence is critical to realizing these dynamic capabilities. Compiled java code (class files) can move to a variety of platforms and be executed without loss of meaning. This powerful capability, which has not been realized to the same level and extent by any other language efforts, was first made generally available by Java implementations.

Java's reflection capabilities, allow a program to discover and access the properties available in an object. This allows internet available objects to be manipulated in a manner not necessarily know by the program at the time it was compiled. In addition to facilitating distribution, Serialization, RMI, and Reflection are leading to a view of internet enabled software service objects. These provide the critical infrastructure for e-commerce services, such as financial, investment, and retail purchase.

The current e-commerce solutions utilize the web and represent a composition of diverse technologies:

1. **User interface** through html, xml, Java, JavaScript, Flash and so on,
2. **Server functions** through dynamic html, JSP, ASP, J2EE, CGI or servlets,
3. **Legacy data** through RMI, ODBC or JDBC connectivity to a relational database.

The challenge is to formulate a secure impenetrable application in light of the combination of a variety of technologies and capabilities.

The Java model for securing the operations in an executing virtual machine has progressed significantly since the initial introduction of the sandbox model for Java applications. Initial versions of Java provided full trust to classes loaded locally and prohibited all sensitive operations from any code obtained dynamically.

Java now supports 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 includes a set of permissions together with a codebase and an indication of who signed the code. 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.

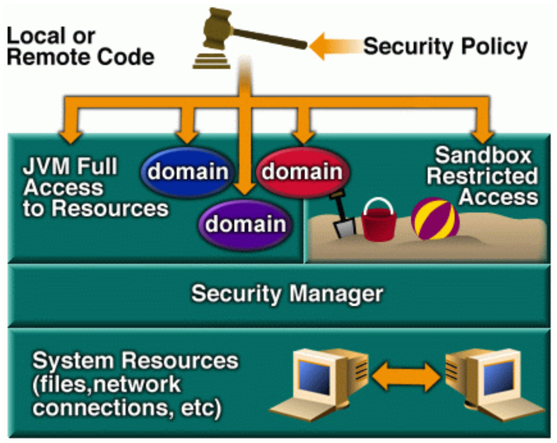

Local or Remote Code
Security Policy
JVM Full Access to Resources
domain
domain
domain
Sandbox Restricted Access
Security Manager
System Resources (files,network connections, etc)

FIGURE 4. Controlling Access to Java Resources

Figure 4, is taken from the On-line Java Tutorial,

http://java.sun.com/docs/books/tutorial/

and shows the interaction between the security domains defined in Java2 and the original sandbox model. In the Java 2 SDK version 1.4, the standard platform has been further augmented to integrate the Java authentication and authorization service (JAAS). Doing so takes a step closer to integrating user login services with authentication mechanisms. See:

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

In Java 2, security domains are defined by a policy 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.

When a protected operation is attempted, the virtual machine's security manager performs a security check. It looks at the classes of all methods currently on the runtime call stack. Each associated protection domain is queried to determine whether the operation is allowed. An operation is performed only if all methods on the runtime stack have the appropriate permission.

Signing executable is an important application of authentication and integrity technology. As the number of distributed applications grows and those applications increasingly rely upon migration of code, we need assurance that we are granting permissions to trustworthy code.

Today, code signing is largely platform dependent. For example, applets executed with the Netscape or Internet Explorer Java virtual machines require use of Netscape or Microsoft tools to sign the code. Applets designed to run with the SUN plug-in virtual machine must be signed with the SUN tools. This lack of consistency only accentuates the problems arising from utilizing multiple technologies to realize an e-commerce application.

5. Cryptography

The SUN implementation of the Java Cryptography Architecture (Java Crypto Extensions) is freely available for developing applications that rely on encryption. Similarly, if the application requires an encrypting web server, Apache-SSL is one of the freely available web servers based on OpenSSL. It can be freely obtained and used commercially. See:

http://www.apache-ssl.org/

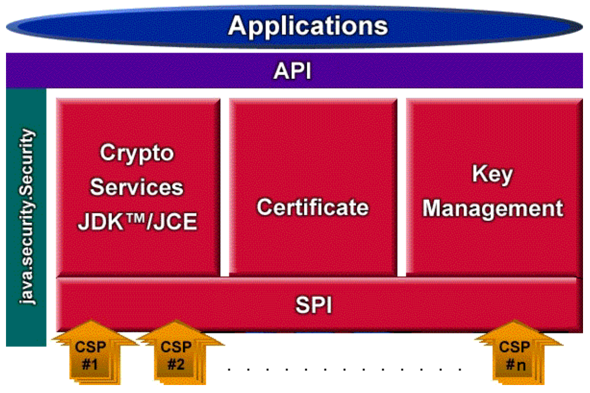
In addition to authentication and integrity, distributed e-commerce applications require cryptographic services. Authentication and integrity assure the identity of the sender and that information was not changed in transmission, but they do not protect against reading during transmission. Encryption is a concern for financial transactions or other communication where personal identification information must be transmitted. To guard against this type of intrusion, many encryption / decryption algorithms and implementations exist.

Encryption is the process of taking clear text and converting it into cypher data that is unreadable to anyone who does not know the key. Decryption reconstructs clear text from the cypher data, using the key. The sender performs encryption before transmission and the receiver decrypts to reconstruct the information. Several implementations of various strength exist. Primary issues relate to strength and performance - time and space to encrypt and decrypt.

Figure 5 is taken from the Java Tutorial and shows the service provider architecture that is used in the security frameworks provided by SUN with Java. The API (application program interface) provides a common interface for developing e-commerce or other distributed applications. To the extent possible, various alternative approaches to security are cast into a single interface. Engine classes abstractly define cryptographic services.

Providers (implementing security services) write to the lower level SPI (service providers interface). For an example implementation see JCSI [5]. SUN also provides a default implementation which is distributed with the downloadable extensions. Where multiple implementations exist, initialization methods select the appropriate implementation based on parameters. This same approach is used, for example in Java's database connectivity, JDBC. Where multiple drivers exist, selection is wired-into the API through initialization methods.

Although this architecture is a powerful approach that adds considerable value to the Java framework, in practice it is very difficult to achieve a single common interface that works equally well for all implementations.



IGURE 5. Service Provider Architecture

6. Closing Remarks

For a language that has developed and whose use has spread so rapidly, Java's features are remarkably complete and consistent. Nevertheless, security in Java applications is a difficult task. Java security mechanisms are complex and as such are likely to be inappropriately used by developers. The Java security model, together with the Java cryptography architecture are powerful tools that are integrated well into the language both in terms of controlling applications and in terms of defining security frameworks that are amenable to realization by multiple implementations.

Authenticity and integrity are just that and no more. Signed Java can be relied upon regarding who signed it and that it has not been disturbed in transmission. The fact that a digital signature has been verified tells the user nothing about the goodness of the code or the security of the system that is delivered in signed form. These are elements of trust in the individual or company that signed the code.

To further the problem, security problems do and will continue to result from problems in the infrastructure upon which the Java implementation is built. For example, denial of service attacks, file access, host system intrusion and underlying problems with TCP/IP all arise to the applications built on these technologies.

Nevertheless, e-commerce applications must be secure and the best way to build in security is to use best software practices and processes for their development. Specification and design of a secure distributed Java application should include security risks, requirements and underlying constraints. Development should proceed with a security risk assessment, followed by design and reviews from risk perspective. Security testing, which is necessarily different from specification testing, should consider likely avenues of problems and exercise documented successful attacks on similar systems.

For further reading on security problems with Java and related technology, see:

http://www.cigital.com/javasecurity/articles-1.html
http://www.w3.org/pub/Conferences/WWW4/Papers/197/40.html

and the Java security website:

http://www.rstcorp.com/java-security.html

For further reading in Security and Encryption, see Peter Guttmann's web site [4], which contains references to various research publications as well as software and other internet resources related to security and encryption.

7. References

[1.] McGraw, Gary and Felton, Ed; Securing Java, John Wiley and Sons Inc., 1999, see: **http://www.securingjava.com/**

[2.] Griscom, Daniel; Code Signing for Java Applets; see: **http://www.suitable.com/Doc_CodeSigning.shtml**

[3.] Campione, Mary, et al., The Java Tutorial, SUN, Addison Wesley, December 2000, **http://java.sun.com/docs/books/tutorial/**

[4.] Guttmann, Peter; Security and Encryption-Related Resources and Links, **http://www.cs.auckland.ac.nz/~pgut001/links.html**

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