Interested in learning more about security?

SANS Institute
InfoSec Reading Room

This paper is from the SANS Institute Reading Room site. Reposting is not permitted without express written permission.

Securing Server Side Java

Java has many security features built in that were needed for Java Applets to protect clients from malicious programmers, one example being the "sandbox" that the Java Virtual Machine utilizes. These feature that provided security to protect users from code downloaded off the Internet can help make server side Java more secure. Java code executed on the Java Virtual machine is different from a typical "C" application running on top of the machine's operating system. Java code running in the virtual machine is restricte...

Copyright SANS Institute
Author Retains Full Rights

AD
Take control of vulnerability threats Now!
Request a Free Trial
Secunia Stay Secure

William Rushmore
GIAC Security Essentials
Assignment Version 1.4b/Option 1

Securing Server Side Java

Introduction

The Java platform began as applets running in client's web browsers and promised to change the Internet. Java captured the interests of many in the computer industry for its ability to "write once, run anywhere." The reality of the "Write once, run anywhere" marketing slogan did not quite live up to the hype. Although Java was and still is a good solution for cross platform client applications, it did not revolutionize client side applications over the Internet. However, one place that Java has made great strides has been with server side applications. This began when Sun Microsystems released the Java Servlet specification. Java Servlets became popular as a more secure and robust alternative to CGI. Since then, Sun has released Java 2, Enterprise Edition (J2EE) that is a specification for an enterprise-class server-centric Java platform. This document intends to provide methods and best practices to secure Server Side Java on the J2EE platform.

The Java Platform

In the early days of Java several flaws affecting security were discovered in the design and with some of the implementations of the Java Virtual Machine Specifications. The good news is that none of these previous issues can be exploited with server side Java. Java has many security features built in that were needed for Java Applets to protect clients from malicious programmers, one example being the "sandbox" that the Java Virtual Machine utilizes. These feature that provided security to protect users from code downloaded off the Internet can help make server side Java more secure. Java code executed on the Java Virtual machine is different from a typical "C" application running on top of the machine's operating system. Java code running in the virtual machine is restricted from accessing resources on the machine outside of the Sandbox as the file system or network resources can only be accessed if explicit permission is given. The buffer overflow is one of the most exploited security flaws of networked applications. The design of the Java Virtual Machine is theoretically immune to such an attack. The Java platform has also benefited from not having a famous and widespread exploit publicized widely. Because of this, some server side Java programmers think that these built in protections are adequate for securing their applications. Nothing could be further from the truth. These protections can only be considered one layer of the security onion.

The Java platform has made it very easy to write networked applications. Because of this, Java has become popular for building dynamic Internet sites and

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

for enterprise's critical distributed applications, increasing the importance of securing server side Java.

Infrastructure

Before one even considers securing server Java applications, the infrastructure that the applications will run on top of must be secure. Without adequate security for the network and the server themselves the efforts to secure Java applications could be circumvented. Securing the server and the network is beyond the scope of this document. Refer to the appropriate documents on secure the operating system and network architecture.

Setting up Java application servers for Internet clients requires more attention than for an intranet client when building the infrastructure. Almost without exception Internet clients interact with server side Java through a web server, whether it is from a web browser or through a Web Service. Refer to appropriate documents for securing your chosen web server.

Operating Systems

The Java Application server process should only be given rights to what the application is required. Do not run the Java processes as "root", "Administrator", or any kind of super user. This is particularly important for file system permissions since this is the most common way Java developers will interact directly with the operating system. (Not a good practice to do this however, see below.)

Some operating systems provide a Java Virtual Machine (JVM) with the operating system. Even most versions of Microsoft Windows have a JVM built in. The operating system patches would normally provide security updates for the JVM. Be aware of this when testing operating system patches if the built in JVM is used.

Application Servers

More than likely, a server side Java application would be deployed to either a commercial or open source application server. Examples would be Servlet engines like Tomcat or JRun and EJB containers like JBoss. Or large commercial applications servers with several components including EJB containers and Servlet engines: Web Logic, Web Sphere, and Sun One. Application servers are the implementation of the J2EE specification. J2EE application servers typically are the middle tier between the web server or client and the database. Functionality varies between application servers but they typically provide Servlet/Java Server Pages (JSP) and Enterprise Java Bean (EJB) containers. Servlets and JSP's are dynamically built web pages, similar in functionality to CGI. Servlets/JSP's are a more secure alternative than CGI since they run on top of the managed Java platform. EJB's are distributed objects, similar to CORBA. The EJB specification has some interesting requirements that

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

aid in creating secure applications. For example, EJB objects may not use file IO and may not start new threads. With extensions such as JAAS (see below) writing more secure applications is even easier.

From a security prospective, the application server should be viewed as one would view an operating system. It is the platform on which your applications will run on top of. Often when selecting an application server price, performance, and ease of use are big but also keep security in mind. Check to see how the vendor or, in the case of open source, the development group has dealt with past security flaws. The typical security web sites are a good start. "The Server Side", www.theserverside.com, is an excellent source for J2EE specific security issues.

Once an application server is chosen and deployed, it must be kept up to date, just like an operating system. Java application servers are not as mature as operating systems so it even more important to test any patches before deploying the patch to a production environment. This author has experienced problems deploying a patch to fix a critical security flaw on a particular commercial application server/web server. The problem occurred when a web client would make ten invalid requests in a short period of time the application server would crash. When this was discovered on a production E-Commerce site, a patch was already available from the vendor and was deployed to the production site right away. However, after some time it was discovered that the patched introduced a bug the broke the functionality of the site for some customers. It took two more patch later until it fixed the fatal security flaw and did not break the website functionality. By this time, the team learned to thoroughly test the patch before deploying to the live site.

Databases

Typically an application server will need to have to access to a dedicated database server. Database access is through an implementation of the Java Database Connectivity API or JDBC. Just like application servers, vendors have their own implementation of the JDBC specification. These JDBC drivers have varying levels of security. Choose a JDBC driver from a trusted vendor and one that provides the security features required by local policy and application requirements.

Avoid allowing clients to connect directly to the database. A proxy on the application server should make connections rather than allowing the client to manage database connections. An object on the applications server typically abstracts this, a typical use of an EJB. Along with performance benefits there are also several security issues this addresses. First, it simplifies authentication. Only one "client", the application server, to the database has to be created and maintained. Second, if only the designated application servers are permitted to connect to the database then network layer can be secured by using such things as trusted hosts or private switched networks, further increasing security. Where this breaks down is when an audit trail is needed on elements of data of what specific user made modifications. Some database vendors provide mechanisms

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

pass in a user connection though the application servers connection. The user's connection is "tunneled" through the application server's connection, so to speak.

Ideally, when data is passed via JDBC from the database to the application the network should be secured to prevent an unauthorized individual from snooping this data, i.e. switched subnet. However, in some situations this is not possible, such as over the Internet. But the risk involved with the disclosure of sensitive may also require additional security even if both the Java application and database are located behind a common firewall on a switched network. In these cases choose a JDBC driver with encryption capabilities. These JDBC drivers will encrypt and data calls, logons, etc. between the application server and the database. If a JDBC call for some reason must be over the Internet then an encrypted driver should always be used. Keep in mind that there will be a performance penalty when encryption and plan for scaling accordingly.

A properly setup database schema is essential. Since users are not directly connecting to the database, sometimes schema security is not addressed as seriously. The database administrator should only give permissions to the required tables and operations needed by the application. In corporate environments, disparate applications sometimes share the same application server and database. As another layer of security, make sure each application has its own connection and user name and password. Also ensure the application user cannot read or modify data is not required.

Secure Code

The most overlooked and by far the most difficult aspect of creating secure Java applications is writing secure Java code. Unfortunately, it would also be the most likely means an attacker would gain unauthorized access. In regards to writing secure code, Java is easier than its predecessors like C and C++ that run on "unmanaged" platforms. Unmanaged applications have almost full access to the underlying memory and operating system. If not coded properly memory leaks and overruns are possible. This can lead to a common security flaw, the dreaded buffer over flow. Because Java is running inside a managed container, a Java developer cannot write code that is vulnerable to a buffer overflow, in theory. The only way a buffer overflow could be exploited would be a flaw in the implementation of the virtual machine, outside of the developer's control. Incidentally, this is another reason why keeping up to date with vendor patches for the application server/JVM is important. However, this should not allow developers to be lulled into a false sense of security when writing Java applications.

When Java first came out, it was marketed as a platform that allowed developers to "Write Once and Run Anywhere." The premise was to allow code to be executed remotely on client's machines over the Internet. For this even to be considered to secure it had to be implemented in a way that would prevent malicious programmers from causing harm and stealing data from client's machines. Some of the security was addressed by providing "sandboxes" that will not allow actions that where not be explicitly allowed by predefined rules.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

There are also such things as mechanisms built into the class loader that prevents code from circumventing the sandbox. All of the following issues that developers should be aware of for writing secure code assume that everything is residing within the sandbox and pertaining to potential problems within the Java Virtual Machine and not exploits of the Java Virtual Machine.

Java is an object-oriented language. One of the fundamental principles of an object-oriented language is to promote reusability. This means that often programmers are writing code or objects to be used by other programmers. Programmers should be mindful of this in design to prevent other programmers from using their code in such a way that could cause security vulnerabilities.

Java passes all values by reference. This is analogous in C to passing a pointer in a method parameter. Therefore the caller of a method could change a variable that is considered private. See code example 1. The code example is made up of three class files. The data class, "Data", the "TrustingProvider" which provides public callers to the object its data object, and notice that the "EvilCaller" class changes its local copy of the "Data" which in turn changes the "TrustingProvider's" local copy of the data. The sample output is as follows:

```
Trusting Provider's Data value is: 1
Trusting Provider's Data value is: 5
```

This example uses a custom built object that is mutable. But this also can be exploited with arrays, even if the arrays contain immutable objects such as Strings.

There are two ways this can be fixed. The first would be to change the Data object so that it immutable. In other words only allow the "privateNumber" to be set when the object is created but never changed during the life of the object. In this example, Data would not have a method called "setNumber". But this may not be practical if the Data object had several data fields or the object needs to be updated during its life cycle.

The second approach would be to do a "defensive copy". This can be done by the "TrustingProvider" returning a copy of "Data". That way "EvilCaller" can make all kinds of changes to its local copy of "Data" and "TrustingProvider's" copy will stay the same. See example 2 for the fixed code. As you can see from example 2's output:

```
Trusting Provider's Data value is: 1
Trusting Provider's Data value is: 1
```

The Java World article Twelve Rules for Developing More Secure Java Code by Gary McGraw and Edward Felton, has several rules that are particularly important for developers writing server side Java code. The first is "Make everything final (unless there's a good reason not to)". Without going into the details object oriented principles, another aspect of Java is the ability of objects to "extend" the functionality of other objects. Children objects also have the ability to override or modify the parent's functionality. A parent object can prevent this by using the key word "final". This can be used to prevent a class

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

from being extended or it can be used to prevent a method from being overridden. Developers should consider using "final" on sensitive classes or methods. This can be a delicate balancing act between writing secure code and promoting reusability. Keep in mind future uses of the code being written and damage that could potentially be done if a class was extended and the damage that could be done. Some good candidates of code that should always be final are ones that represents business logic or any custom-built security logic.

Another rule is "Limit access to your classes, methods, and variables." This relates to the above secure code guidelines. The article states, "Every class, method, and variable that is not private provides a potential entry point for an attacker". Notice that the "Data" class in example 1 and 2 has its variable "private". Only methods needed by external callers are made public and everything else should be explicitly classified as "private".

Related to the above rule is Rule 4, "Don't depend on package scope." If a class, method, or variable is not labeled as "public", "private", or "protected" it can be accessed by any code within the same package. Most Java virtual Machines have no mechanism to keep someone from inserting their own code into someone else's packages (the exception being the platform's "java.lang" package). The only way to prevent this on those Java Virtual Machines is through mechanisms at deployment. If security is a concern use "private" and don't rely on the package scope control mechanisms, which are designed for enforcing software-engineering principles or trust it will always be deployed correctly.

Rule 9 and 10 relate to dangers of serialization. Basically when an object implements "Serializeable" it has the ability to be passed outside of the Java Virtual Machine. Typical uses are sending Java objects to other Java platforms over a network or saving them to files to be used later. While the serialized object is outside of the JVM there is no control of the object. The objects non-encrypted data can easily be read. The Twelve Rules for Developing More Secure Java recommends avoiding serialized objects altogether. But the Sun secure code guidelines web page has some tips for securing serialized objects. Some important principles are to keep any handles to resources such as file handles declared as "transient" (the transient keyword instructs the JVM not to serialize the variable). Communications that transport serialized data should utilize encryption.

Java has a feature called Java Native Interface (JNI), which allows the Java Virtual Machine to call methods on the underling operating system. Server side Java programs will rarely ever require the use of JNI and should be avoided. However, if JNI is used, the code called should be scrutinized for security thoroughly. Refer to the Sun secure code guidelines in the references for things to look for.

A similar issue is the use of "Runtime.getRuntime().exec()" command. This allows the Java virtual machine to fork process on the operating system. Obviously the forked process will be outside of the controls provided by the Java platform. With the rich API provided by the Java platform this should rarely be needed. Avoid this when writing server side Java.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Direct file IO from server side applications should also be avoided. An attacker could exploit other vulnerabilities and read data from sensitive files, such as password files, etc. If data needs to be persisted use directory servers, databases, or a properties file. If file IO must be used to read properties file, ensure that JVM process has the minimum permissions necessary.

Another aspect of the Java Platform is the concept of automatic garbage collection. This is great from a developer's perspective and helps avoid problems such as memory leaks. But this means that when an object is out of scope it still exists until the garbage collection process de-allocates it. So the object will exist in the heap potentially with its sensitive data. The object could be around for a very long time or even never be removed. Therefore, when an object is no longer needed and ready to be sent to the garbage collector, all sensitive data should be cleared if possible. This makes a heap-inspection attack more difficult to carry out.

The Java platform has a very rich set of libraries available to programmers and this continues to grow with every new release. Many features available can help developers with creating secure code. The latest release of the Java API (JDK 1.4.1 at the time of writing) contains such things as cryptography, Secure Sockets, and mechanisms for authentication and authorization. When designing applications the built in mechanisms should always be used before implementing any "home grown" solutions. The Java API's will have the scrutiny of the entire community and will intrinsically be more secure.

One of Java's features is the Reflection API. This API allows programmers to view representations of classes and objects on the VM. Tool builders use this API for things such as debuggers, class browsers, and GUI builders. Using reflection has negative impacts on performance and security, especially invoking methods using reflection. Business programmers writing server side Java should never use reflection.

Secure Communication

When clients communicate to Java Server Applications there are three ways this is typically done. First is through HTTP. This is especially true for Internet clients outside of the firewall. It can be a human through a web browser or a process like a Web Service client. Typically a web server such as Apache or Iis will provide the HTTP communications for the Java application server. Many documents have been written on securing various web servers; refer to the appropriate document for your web server. Obviously, these communications that require security should utilize the popular HTTPS protocol.

Another common method for Java clients to communicate with Java server applications is through Java Remote Method Invocation (RMI). With RMI it is very easy to build distributed applications but the problem is that security such as encryption and authentication/authorization are not built into the protocol. So any security must be custom built. RMI is the protocol used for Enterprise Java Bean (EJB) communication. Refer to the JAAS section below on how some EJB containers are using JAAS to address authentication/authorization issues.

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Smaller scale server side Java applications may use lower lever socket communication. Just like RMI, the developer must build the authentication and authorization. However, the most recent versions of the Java platform have the ability to use SSL to help protect against someone from "snooping" sensitive data.

JAAS

A relatively new technology for securing Java is the Java Authentication and Authorization Service (JAAS). All of the other Java mechanisms where designed to protect from malevolent programmers. JAAS is intended to provide a mechanism for Java applications to providing authorization and authentication with little effort on the part of application programmers. As stated above, for a more secure application, don't rewrite things you can use from somewhere else.

JAAS began as an extension to the Java platform and with the release of Java 2 Standard Edition version 1.4 JAAS became integrated into the Java platform. JAAS is also part of J2EE 1.3. The commercial application servers are just beginning to support JAAS and J2EE 1.3.

JAAS uses what is called a Pluggable Authentication Module (PAM) framework. The PAM framework has been used on other platforms such as Solaris. Using this framework simplifies building applications because it can be independent of authentication mechanism. As a matter of fact, the authentication module is specified in properties files so this could be changed without modifying the Java source code. Authentication modules exist for use against directory servers, UNIX, Windows, and Kerberos. Applications are not limited to just using one module. The developer must implement the appropriate authentication modules for their environment.

In addition to writing the authentication module, a callback handler must be implemented. The callback handler is the mechanism passes authentication data to the server. The good news is that there is a high level of reusability with the authentication module and the callback handler. These pieces of code would probably be only written once at an organization or obtained from a third party source.

Once the authentication module and callback handler is completed, most of the work in utilizing JAAS is not from the application developer but from release engineer or system administrator deploying the JAAS application. There are a few properties files that must be configured specific to JAAS. The first and easiest is the login configuration file. This file lists the login module or modules to be used to authenticate users. The other file is the policy file. The policy file is where the rules will be setup for authentication and authorization. To help build this file a graphical policy editor is provided with the Java 1.4 SDK. Here is a sample of what the policy file looks like:

```
/** Subject-Based Access Control Policy for the JAAS
Sample Application **/
```

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

```
grant     codebase
"file://D:/data/jaas/jaas1_0/doc/sample/sample_action.
jar",
     Principal sample.SamplePrincipal "testUser" {


   permission java.util.PropertyPermission
"java.home", "read";
   permission java.util.PropertyPermission
"user.home", "read";
   permission java.io.FilePermission "foo.txt",
"read";
};
```

In conclusion, securing a server side Java application has many facets: from securing the network and underlying servers, to writing secure code. This compilation of guidelines should by no means be considered a comprehensive list but summation of important points to help the give the reader a better understanding of what is involved with securing server side Java applications.

© SANS Institute 2003, Author retains full rights

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

References:

Java: Potent Security:
http://www.eweek.com/article2/0,3959,5404,00.asp

How JAAS enables use of custom security repositories with J2EE applications:
http://www.theserverside.com/resources/article.jsp?l=Pramati-JAAS

Java Security FAQ:
http://www.cs.princeton.edu/sip/faq/java-faq.php3

JDBC Drivers and Web Security:
http://www.ddj.com/documents/s=918/ddj9807j/9807j.htm

12 Rules of More Secure Java code:
http://www.javaworld.com/javaworld/jw-12-1998/jw-12-securityrules.html

Security Code Guidelines:
http://java.sun.com/security/seccodeguide.html#gcg

Making Defensive Copies of Objects:
http://developer.java.sun.com/developer/JDCTechTips/2001/tt0904.html#tip1

Maximum Server Security:
http://www.fawcette.com/Archives/premier/mgznarch/javapro/2001/06jun01/jsw0106/jsw0106-3.asp

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

© SANS Institute 2003, As part of the Information Security Reading Room. Author retains full rights.

Example 1

```
public class Data
{
      private int privateNumber;

      public Data(){}

      public Data(int numberIn)
      {
            privateNumber = numberIn;
      }

      public int getNumber()
      {
            return privateNumber;
      }

      public void setNumber(int numberIn)
      {
            privateNumber = numberIn;
      }
}


public class TrustingProvider
{
      private static Data myData = new Data();

      public TrustingProvider()
      {
            myData.setNumber(1);
      }

      public Data getDataObject()
      {
            return myData;
      }

      public void printYourData()
      {
            System.out.println("Trusting Provider's Data value is: " +
myData.getNumber());
      }
}

public class EvilCaller
{
      public static void main(String[] args)
      {
            TrustingProvider chump = new TrustingProvider();
            chump.printYourData();
            Data localCopy = chump.getDataObject();
            localCopy.setNumber(5);
            localCopy = null;
            chump.printYourData();
```

© SANS Institute 2003, Author retains full rights

```
        }
}
```

© SANS Institute 2003, Author retains full rights

Example 2

```
public class Data
{
      private int privateNumber;

      public Data(){}

      public Data(int numberIn)
      {
            privateNumber = numberIn;
      }

      public int getNumber()
      {
            return privateNumber;
      }

      public void setNumber(int numberIn)
      {
            privateNumber = numberIn;
      }
}

public class TrustingProvider
{
      private static Data myData = new Data();

      public TrustingProvider()
      {
            myData.setNumber(1);
      }

      public Data getDataObject()
      {
            Data rtnValue = new Data(myData.getNumber());
            return rtnValue;
      }

      public void printYourData()
      {
            System.out.println("Trusting Provider's Data value is: " +
myData.getNumber());
      }
}

public class EvilCaller
{
      public static void main(String[] args)
      {
            TrustingProvider chump = new TrustingProvider();
            chump.printYourData();
            Data localCopy = chump.getDataObject();
            localCopy.setNumber(5);
            localCopy = null;
            chump.printYourData();
```

© SANS Institute 2003, Author retains full rights

```
        }
}
```

© SANS Institute 2003, Author retains full rights

Last Updated: May 23rd, 2013

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS Malaysia @ MCMC 2013	Cyberjaya, MY	Jun 03, 2013 - Jun 08, 2013	Live Event
SANS Pen Test Berlin 2013	Berlin, DE	Jun 03, 2013 - Jun 08, 2013	Live Event
Industial Control Systems Security Training - Houston	Houston, TXUS	Jun 10, 2013 - Jun 15, 2013	Live Event
Security Impact of IPv6 Summit 2013	Washington, DCUS	Jun 14, 2013 - Jun 16, 2013	Live Event
SANSFIRE 2013	Washington, DCUS	Jun 14, 2013 - Jun 22, 2013	Live Event
SANS Canberra 2013	Canberra, AU	Jul 01, 2013 - Jul 13, 2013	Live Event
Digital Forensics & Incident Response Summit 2013	Austin, TXUS	Jul 09, 2013 - Jul 16, 2013	Live Event
SANS London Summer 2013	London, GB	Jul 09, 2013 - Jul 16, 2013	Live Event
SANS Rocky Mountain 2013	Denver, COUS	Jul 14, 2013 - Jul 20, 2013	Live Event
SANS Mumbai 2013	Mumbai, IN	Jul 22, 2013 - Jul 27, 2013	Live Event
SEC528 SANS Training Program for the CompTIA® New Advanced Security Practitioner (CASP) Certification	Washington, DCUS	Jul 22, 2013 - Jul 26, 2013	Live Event
SANS San Francisco 2013	San Francisco, CAUS	Jul 29, 2013 - Aug 03, 2013	Live Event
SANS SEC 560: Network Penetration Testing @ Bangalore 2013	Bangalore, IN	Aug 05, 2013 - Aug 10, 2013	Live Event
SANS Boston 2013	Boston, MAUS	Aug 05, 2013 - Aug 10, 2013	Live Event
Critical Security Controls Summit	Washington, DCUS	Aug 12, 2013 - Aug 18, 2013	Live Event
Industrial Control Systems Security Training - DC	Washington, DCUS	Aug 12, 2013 - Aug 16, 2013	Live Event
SANS Thailand 2013	Bangkok, TH	Aug 19, 2013 - Aug 31, 2013	Live Event
SANS Virginia Beach 2013	Virginia Beach, VAUS	Aug 19, 2013 - Aug 30, 2013	Live Event
Mobile Device Security Summit 2013	OnlineCAUS	May 30, 2013 - Jun 06, 2013	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced