Software Vulnerabilities in Java

Fred Long

October 2005

CERT

Technical Note
CMU/SEI-2005-TN-044

This work is sponsored by the U.S. Department of Defense.

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Contents

Abstract

Java is essentially a safe language with good security features. However, there are several Java features and facilities that can compromise safety if they are misused or improperly implemented. This report briefly describes these potential software vulnerabilities in the current version of Java, Java 5.

1 Introduction

This brief report is concerned with software vulnerabilities in the current version of Java, that is, Java 5.

Java is essentially a safe language: there is no explicit pointer manipulation; array and string bounds are automatically checked; attempts at referencing a null pointer are trapped; the arithmetic operations are well defined and platform independent, as are the type conversions. The built-in `bytecode verifier` ensures that these checks are always in place.

Moreover, there are comprehensive, fine-grained security mechanisms available in Java that can control access to individual files, sockets, and other sensitive resources. To take advantage of the security mechanisms, the Java Virtual Machine (JVM) must have a `security manager` in place. This is an ordinary Java object of class `java.lang.SecurityManager` (or a subclass) that can be put in place programmatically but is more usually specified via a command line parameter.

There are, however, some ways in which Java program safety can be compromised. These are described in Section 2.

2 Areas of Potential Vulnerability

2.1 Type Safety

Java is believed to be a type-safe language [LSOD 02, Sec. 5.1]. Hence, it should not be possible to compromise a Java program by misusing the type system. To see why type safety is so important, consider the following types:

```
public class TowerOfLondon {
   private Treasure
      theCrownJewels;
   ...
}
```

```
public class GarageSale {
   public Treasure
      fredsJunk;
   ...
}
```

If these two types could be confused, it would be possible to access the private field `theCrownJewels` as if it were the public field `fredsJunk`. More generally, a "type confusion attack" could allow Java security to be compromised by making the internals of the security manager open to abuse. A team of researchers at Princeton University showed that any type confusion in Java could be used to completely overcome Java's security mechanisms (see *Securing Java* Ch. 5, Sec. 7 [McGraw 99]).

Java's type safety means that fields that are declared `private` or `protected` or that have default (package) protection should not be globally accessible. However, there are a number of vulnerabilities "built in" to Java that enable this protection to be overcome. These should come as no surprise to the Java expert, as they are well documented, but they may trap the unwary.[1]

2.2 Public Fields

A field that is declared `public` may be directly accessed by any part of a Java program and may be modified from anywhere in a Java program (unless the field is declared `final`). Clearly, sensitive information must not be stored in a public field, as it could be compromised by anyone who could access the JVM running the program.

[1] Vulnerabilities described in this technical note have only been evaluated for Java 5. The code used in testing was executed using Java version 1.5.0_04 on a Windows XP system.

2.3 Inner Classes

Inner classes have access to all the fields of their surrounding class. There is no bytecode support for inner classes, so they are compiled into ordinary classes with names like `OuterClass$InnerClass`. So that the inner class can access the private fields of the outer class, the private access is changed to package access in the bytecode. Hence, hand-crafted bytecode can access these private fields (see "Security Aspects in Java Bytecode Engineering" [Schönefeld 02] for an example).

2.4 Serialization

Serialization enables the state of a Java program to be captured and written out to a byte stream [Sun 04b]. This allows for the state to be preserved so that it can be reinstated (by deserialization). Serialization also allows for Java method calls to be transmitted over a network for Remote Method Invocation (RMI). An object (called `someObject` below) can be serialized as follows:

```
ObjectOutputStream oos = new ObjectOutputStream (
          new FileOutputStream ("SerialOutput") );

oos.writeObject (someObject);
oos.flush ( );
```

The object can be deserialized as follows:

```
ObjectInputStream ois = new ObjectInputStream (
         new FileInputStream ("SerialOutput") );

someObject = (SomeClass)ois.readObject ( );
```

Serialization captures all the fields of a class, provided the class implements the `Serializable` interface, including the non-public fields that are not normally accessible (unless the field is declared `transient`). If the byte stream to which the serialized values are written is readable, then the values of the normally inaccessible fields may be read. Moreover, it may be possible to modify or forge the preserved values so that when the class is deserialized, the values become corrupted.

Introducing a security manager does not prevent the normally inaccessible fields from being serialized and deserialized (although permission must be granted to write to and read from the file or network if the byte stream is being stored or transmitted). Network traffic (including RMI) can be protected, however, by using SSL.

2.5 Reflection

Reflection enables a Java program to analyze and modify itself. In particular, a program can find out the values of field variables and change them [Forman 05, Sun 02]. The Java reflection API includes a method call that enables fields that are not normally accessible to be accessed under reflection. The following code prints out the names and values of all fields of an object `someObject` of class `SomeClass`:

```
Field [ ] fields = SomeClass.getDeclaredFields ( );

for (Field fieldsI : fields) {

   if ( !Modifier.isPublic (fieldsI.getModifiers ( )) )
   {
      fieldsI.setAccessible (true);
   }

   System.out.print   (“Field: “ + fieldsI.getName ( ));
   System.out.println (“, value: “ +
                       fieldsI.get (someObject));

}
```

A field could be set to a new value as follows:

```
String newValue = reader.readLine ( );
fieldsI.set (someObject,
        returnValue (newValue, fieldsI.getType ( )) );
```

Introducing the default security manager does prevent the fields that would not normally be accessible from being accessed under reflection. The default security manager throws `java.security.AccessControlException` in these circumstances. However, it is possible to grant a permission to override this default behavior: `java.lang.reflect.ReflectPermission` can be granted with action `suppressAccessChecks`.

2.6 The JVM Tool Interface

Java 5 introduced the JVM Tool Interface (JVMTI) [Sun 04d], replacing both the JVM Profiler Interface (JVMPI) and the JVM Debug Interface (JVMDI), which are now deprecated.

The JVMTI contains extensive facilities to find out about the internals of a running JVM, including facilities to monitor and modify a running Java program. These facilities are rather low level and require the use of the Java Native Interface (JNI) and C Language programming. However, they provide the opportunity to access fields that would not

normally be accessible. Also, there are facilities that can change the behavior of a running Java program (for example, threads can be suspended or stopped).

The JVMTI works by using agents that communicate with the running JVM. These agents must be loaded at JVM startup and are usually specified via one of the command line options `–agentlib:` or `–agentpath:`. However, agents can be specified in environment variables, although this feature can be disabled where security is a concern. The JVMTI is always enabled, and JVMTI agents may run under the default security manager without requiring any permissions to be granted. More work needs to be done to determine under exactly what circumstances the JVMTI can be misused.

2.7 Debugging

The Java Platform Debugger Architecture (JPDA) builds on the JVMTI and provides high-level facilities for debugging running Java systems [Sun 04c]. These include facilities similar to the reflection facilities described above for inspecting and modifying field values. In particular, there are methods to get and set field and array values. Access control is not enforced so, for example, even the values of private fields can be set.

Introducing the default security manager means that various permissions must be granted in order for debugging to take place. The following policy file was used to run the JPDS Trace demonstration under the default security manager:

```
grant {
    permission java.io.FilePermission "traceoutput.txt",
        "read,write";
    permission java.io.FilePermission
        "C:/Program Files/Java/jdk1.5.0_04/lib/tools.jar",
        "read";
    permission java.io.FilePermission "C:/Program",
        "read,execute";
    permission java.lang.RuntimePermission "modifyThread";
    permission java.lang.RuntimePermission
        "modifyThreadGroup";
    permission java.lang.RuntimePermission
        "accessClassInPackage.sun.misc";
    permission java.lang.RuntimePermission
        "loadLibrary.dt_shmem";
    permission java.util.PropertyPermission "java.home",
        "read";
    permission java.net.SocketPermission "<localhost>",
        "resolve";
    permission com.sun.jdi.JDIPermission
        "virtualMachineManager";

};
```

2.8 Monitoring and Management

Java contains extensive facilities for monitoring and managing a JVM [Sun 04e]. In particular, the Java Management Extension (JMX) API enables the monitoring and control of class loading, thread state and stack traces, deadlock detection, memory usage, garbage collection, operating system information, and other operations [Sun 04a]. There are also facilities for logging monitoring and management. A running JVM may be monitored and managed remotely.

For a JVM to be monitored and managed remotely, it must be started with various system properties set (either on the command line or in a configuration file). Also, there are provisions for the monitoring and management to be done securely (by passing the information using SSL, for example) and to require proper authentication of the remote server. However, users may start a JVM with remote monitoring and management enabled with no security for their own purposes, and this would leave the JVM open to compromise from outsiders. Although a user could not easily turn on remote monitoring and management by accident, they might not realize that starting a JVM so enabled, without any security also switched on, could leave their JVM exposed to outside abuse.

3 Summary

Java is essentially a safe language with good security features. A review of the US-CERT vulnerability database found no vulnerabilities that were not the result of implementation bugs [US-CERT 05]. *Java and Java Virtual Machine Security* [LSOD 02] and *Securing Java* [McGraw 99] also describe some Java vulnerabilities that have resulted from implementation bugs. However, there are a number of Java features and facilities that an unwary user might not realize could compromise safety.

References

URLs are valid as of the publication date of this document.

[Forman 05] Forman, Ira R. & Forman, Nate. *Java Reflection in Action*. Greenwich, CT: Manning Publications Co., 2005.

[LSOD 02] Last Stage of Delirium Research Group. *Java and Java Virtual Machine Security*. Poland: Last Stage of Delirium Research Group, 2002. http://www.lsd-pl.net/documents/javasecurity-1.0.0.pdf.

[McGraw 99] McGraw, Gary & Felten, Edward W. *Securing Java: Getting Down to Business with Mobile Code*, 2nd ed. New York, NY: John Wiley & Sons, 1999.

[Schönefeld 02] Schönefeld, Marc. "Security Aspects in Java Bytecode Engineering." Blackhat Briefings 2002, Las Vegas, August 2002. http://www.blackhat.com/presentations/bh-usa-02/bh-us-02-schonefeld-java.ppt.

[Sun 02] Sun Microsystems, Inc. *Reflection*. http://java.sun.com/j2se/1.5.0/docs/guide/reflection/index.html (2002).

[Sun 04a] Sun Microsystems, Inc. *Java Management Extensions (JMX)*. http://java.sun.com/j2se/1.5.0/docs/guide/jmx/index.html (2004).

[Sun 04b] Sun Microsystems, Inc. *Java Object Serialization Specification, Version 1.5.0*. http://java.sun.com/j2se/1.5.0/docs/guide/serialization/spec/serialTOC.html (2004).

[Sun 04c] Sun Microsystems, Inc. *Java Platform Debugger Architecture*. http://java.sun.com/j2se/1.5.0/docs/guide/jpda/index.html (2004).

[Sun 04d] Sun Microsystems, Inc. *JVM Tool Interface*. http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/jvmti.html (2004).

[Sun 04e] Sun Microsystems, Inc. *Monitoring and Management for the Java Platform*. http://java.sun.com/j2se/1.5.0/docs/guide/management/index.html (2004).

[US-CERT 05] US-CERT. The US-CERT Vulnerability Notes Database. http://www.kb.cert.org/vuls/ (2005).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	October 2005	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Software Vulnerabilities in Java	FA8721-05-C-0003

6. AUTHOR(S)

Fred Long

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2005-TN-044

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

Java is essentially a safe language with good security features. However, there are several Java features and facilities that can compromise safety if they are misused or improperly implemented. This report briefly describes these potential software vulnerabilities in the current version of Java, Java 5.

14. SUBJECT TERMS	15. NUMBER OF PAGES
vulnerability, computer security, Java, secure programming	16

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL