Global Information Assurance Certification Paper

Java Virtual Machine Security and the Brown Orifice Attack

Miles McQueen

August 14, 2000

Overview

In 1996, Sun Microsystems unleashed a powerful new programming language: Java. Based on networks and the idea that the same software should run on many different kinds of computers, consumer gadgets, and other devices, Java is fundamentally different from many other programming languages[1]. Unlike most other languages, in which it is necessary to compile the source code for each different platform, Java source code is compiled once, into a special type of intermediate code termed byte code. This byte code can then be executed on any device that has a byte code interpreter, or Java Virtual Machine. Most web browsers today have an integrated Java Virtual Machine to run applets, or Java programs that are downloaded from a web server and designed to run in a web browser.

Java Virtual Machine

The Java Virtual Machine is responsible for interpreting byte code or converting byte code into native language code so that the byte code can be executed on the device. The Java VM consists of a byte code interpreter and the core Java classes implementing basic Java functionality. The set of core classes is a little different for each vendor (see Appendix A). The Java VM is also responsible for implementing and applying the Java Security Model[5]. If the Java VM does not implement the Java Security Model exactly and correctly, many security exploits are possible. Given the code size of a typical Java VM (~180,000 lines of C++ code) and the complexity of the Java Security Model, it is highly unlikely that there are zero flaws in any Java Virtual Machine. Indeed, the three most used Java VM's, made by Netscape, Sun Microsystems, and Microsoft, have all had serious security flaws[3,6]. The effects of these flaws have ranged from exploits that crash the browser to some which have completely compromised the client machine.

The Java Security Model

Java Security is built upon the idea that every application runs in a "sandbox." If a Java program attempts to take some action that is outside the "sandbox," a security exception is thrown and the program fails to execute. The three major components of the "sandbox" are the Verifier, the Class Loader, and the Security Manager. The Verifier checks to make sure that the byte code itself follows specific rules that help to assure that the code won't, either by accident or by intent, be able to crash the Java VM. These checks provide one defensive layer that makes the malicious hackers task significantly more difficult. The Class Loader is responsible for ensuring that code from different origin servers do not overwrite each other. This prevents a hostile server from overwriting and thus spoofing any of the core classes on the client machine. In some implementations this would include a special core class, the Security Manager, which is a significant component of the Java VM itself. The Security Manager is responsible for ensuring that Java programs only do what they are allowed to do and no more. For example, an unsigned applet will have no ability to read or write files to a user's system, whereas a fully signed and trusted applet would be able to read and write almost any file. Unfortunately, there currently exists no industry-wide standard for signing Java code and thus no way of giving an applet

extended privileges on all three major Java Virtual Machines without signing the code differently for each Java VM. Each of the three major vendors also has their own methods for granting Java Applets additional functionality. It is for this reason that most downloaded applets, unsigned AND signed, are not allowed to read or write files on a user's system, nor are they allowed to establish connections with servers other than the origin server. Still, applets can be dangerous because of flaws in the Java VM and the core classes. Two of the most recently discovered flaws are exploited by a Java applet called "Brown Orifice."

Brown Orifice

On August 3rd, 2000, a person calling himself Dan Brumleve released a Java Applet with source code that utilizes two distinct security flaws in the Netscape Java Virtual Machine and core classes. His attack applet turns any Netscape version 4.x browser, that downloads and runs the applet, into a web server that anyone online can access[7]. This malicious applet includes the ability to view, modify, and delete any of the files advertised (i.e. visible and accessible) by this new and unintentionally created applet web server[4]. What differentiates this attack from others is that almost 1,000 computers were infected before Netscape issued a security bulletin, and other more dangerous variants were thought to be in use by hackers around the world[5]. It was discovered that the Sun Java Virtual Machine for Java 1 suffered from a security flaw that would allow a malicious user to connect to hosts other than the one that the applet came from[2]. The Java 2 Virtual Machine from Sun Microsystems is not vulnerable because the flaw has been fixed (But have new flaws been added?). It was also determined that the core classes used by Netscape suffered from the second security flaw which allowed files to be read, written, and modified on the client machine. Not all vendor Java VMs and core classes had these vulnerabilities. In particular, the latest Microsoft Virtual Machine was not encumbered with either flaw. What security measures has Microsoft taken that stymies the "Brown Orifice" attack, and what implementation bugs are present in the Netscape and Sun Java Virtual Machines, that allows this "Brown Orifice" exploit to succeed?

The Two Flaws

To postulate an answer to this question, it is necessary to understand the nature of each of the two flaws. The first of the two flaws, which allows an applet to open and close network connections with hosts other than the origin host, from which the applet itself was downloaded, is not strictly speaking a flaw in the virtual machine itself, but rather an implementation bug in the core classes (see Appendix A) supplied with the Netscape and Sun Virtual Machines. According to the Java Security Model, the core classes are supposed to check with the virtual machine's Security Manager before performing any "dangerous" operation[5]. Indeed, all three major Java VMs perform this check and consider opening and closing network connections a "dangerous" operation. The Brown Orifice exploit avoids the Security Manager check by overloading two methods present in the core classes, ServerSocket.open() and Socket.open(). If the two core classes with these methods were implemented correctly, this would be caught by the Security Manager and a security exception thrown, but the core Java classes in the susceptible virtual machines assume that ServerSocket.open() and Socket.open() are trustable and therefore do not need to be checked with the Security Manager, even though these two methods might have been overloaded. This allows an applet to create a ServerSocket or Socket and communicate with hosts other than the origin server as well as allowing the applet to become a server on a local port.

Note that this flaw alone does not allow the server to advertise files on the client system; it simply allows the applet to accept http connections from client machines.

The second flaw, which allows an applet to read/modify/delete files on the client machine, is a flaw in the Security Manager of Netscape's Java Virtual Machine. When a Java Applet attempts to open a URLInputStream or URLConnection, the core class asks the Security Manager if the applet has the privilege to open a connection with the specified URL. When the Netscape Security Manager is presented with a URL specifying a local path that has been constructed in a certain way, the Security Manager incorrectly says that the applet has the privilege to open the connection. This leads to the ability of an applet to read/modify/delete files on the local machine.

These two flaws are used together by Brown Orifice to turn the client browser into an http server that allows almost anyone in the world to read/modify/delete files residing on the client (turned server) machine.

What Does Microsoft's Virtual Machine Do That Prevents Both These Flaws?
For the first flaw, Microsoft's Virtual Machine is protected because the core classes, quite correctly, do NOT trust ServerSocket.open() and Socket.open(). The Security Manager correctly throws the following exception[8]:

```
com.ms.security.SecurityExceptionEx[BOServerSocket.]: cannot access 8080
    at com/ms/security/permissions/NetIOPermission.check
    at com/ms/security/PolicyEngine.deepCheck
    at com/ms/security/PolicyEngine.checkPermission
    at com/ms/security/StandardSecurityManager.chk
    at com/ms/security/StandardSecurityManager.checkListen
    at java/net/ServerSocket.
    at java/net/ServerSocket.
    at BOServerSocket.
    at BOHTTPD.init
    at com/ms/applet/AppletPanel.securedCall0
    at com/ms/applet/AppletPanel.securedCall
    at com/ms/applet/AppletPanel.processSentEvent
    at com/ms/applet/AppletPanel.processSentEvent
    at com/ms/applet/AppletPanel.run
    at java/lang/Thread.run
```

For the second flaw, Microsoft's Virtual Machine is protected because Microsoft's custom Security Manager strictly disallows URL connections to the local machine.

What, If Anything, Is in Place to Prevent Similar Flaws From Being Exploited?
Given the current state of the practice in software development there can be no credible guarantee from any of the vendors that their Java VM or the core classes are implemented 100% correctly. Exacerbating the situation is the inherent complexity of the Java Security Model, which almost guarantees that more, as yet undetected flaws, exist in all implementations. Only continued, diligent searching will weed-out some of the remaining flaws while, hopefully, not introducing

new ones.

Conclusion

"Brown Orifice" shows how simple, small mistakes in the implementation of the large and complex Java Virtual Machine and trusted core classes can lead to serious security holes. If the Java VM and the core classes are not secure then downloaded Java code is not secure, and thus your client machine is not secure. Since it is beyond our capability to guarantee the security of the Java VM and core classes, all of us using browsers and allowing applets to execute are accepting some indeterminate level of threat to having our machines catastrophically compromised.

Acknowledgments

Thanks are due Tyrel McQueen without whom this paper would never have been completed.

References

1. "What is the Java™ Platform?" 19 October 1999. URL: http://java.sun.com/nav/whatis/ (13 August 2000).
2. "Java Security API." 11 Aug. 2000. URL: http://java.sun.com/security/ (13 Aug. 2000).
3. "Current Security Notes." Netscape Security Notes. 8 Aug. 2000. URL: http://home.netscape.com/security/notes/index.html (13 Aug. 2000).
4. Brumleve, Dan. "Brown Orifice HTTPD." 10 Aug. 2000. URL: http://www.brumleve.com/BrownOrifice/BOHTTPD.cgi (12 Aug. 2000).
5. McGraw, Gary and Edward Felten. "Securing Java." New York: Wiley, 1999.
6. "Microsoft VM." Microsoft Technet Security Bulletins. 3 Aug. 2000. URL: http://www.microsoft.com/technet/security/current.asp?productID=23 (13 Aug. 2000).
7. "Experts warn of Netscape security hole." 8 Aug. 2000. URL: http://www.usatoday.com/life/cyber/tech/review/crh387.htm (8 Aug. 2000).
8. "Articles: Java Security Hole Make Netscape Into Web Server." 6 Aug. 2000. URL: http://slashdot.org/comments.pl?sid=00/08/06/0222241&cid=208 (12 Aug. 2000).

Appendix A

Core classes are as follows:

All Three JVM's	Microsoft Specific*	Netscape Specific*	Sun Specific*
java.applet	com.ms.activeX	netscape.plugin	sun.audio
java.awt	com.ms.applet	netscape.net	sun.awt.image
java.awt.datatransfer	com.ms.awt	netscape.javascript	sun.beans.editors
java.awt.event	com.ms.beans	netscape.security	sun.beans.infos
java.awt.image	com.ms.com		sun.io
java.beans	com.ms.debug		sun.misc
java.io	com.ms.dll		sun.net
java.lang	com.ms.fx		sun.net.ftp
java.lang.reflect	com.ms.io		sun.net.nntp
java.math	com.ms.lang		sun.net.smtp
java.net	com.ms.license		sun.net.www
java.rmi	com.ms.net		
java.rmi.dgc	com.ms.object		
java.rmi.registry	com.ms.packagemanager		
java.rmi.server	com.ms.security		
java.security	com.ms.ui		
java.security.acl	com.ms.util		
java.security.interface	com.ms.vm		
java.sql	com.ms.win32		
java.text			
java.util			
java.util.zip			

* The current Microsoft Virtual Machine implements all of the Microsoft Specific packages, but also some (but not all) of the Netscape or Sun Specific packages. Neither the Netscape nor the Sun Virtual Machine implement any of the Microsoft Specific packages.