INTRODUCTION
Why should Java security concern you? Many push-based applications are being ported to Java. In addition, Java is one of the cornerstones of active content and an understanding of Java security basics is necessary for understanding the implications of push security issues.

A lot of people ask: “Why do I need Java security? I thought it was safe.” Java as a language is basically safe and is built on top of a robust security architecture. But security breaches related to bugs in the browser, poorly written Java code, malicious Java programs, poorly written CGI scripts and Javascript code, and more often occur. Moreover, placing the enforcement of a security policy in the browser, and thus in the hands of end users, opens up many opportunities for security measures to be defeated. In addition, many push vendors are relatively new startups that do not always understand mission-critical software and security needs. Such circumstances only exacerbate the security predicament.

While some people might opine that Java is too insecure to be used in production environments and that it should be completely avoided, doing so creates the situation where a tremendous computing opportunity is lost. While the company that decides to bypass Java alleviates itself of Java security worries, that means that they also relinquish the myriad benefits that Java affords. In addition, a significant amount of cutting-edge Internet-based activities such as E-commerce, online trading and banking, and more are all written in Java. Also, many firewall and router vendors are writing their management front-end application in Java. By cutting an organization off from Java, they may likely cut themselves off from the next generation of computing technology.

PAYOFF IDEA
An understanding of Java security basics is necessary for an understanding of the potential risks of push-based applications. This article provides an introduction to Java and then lists some guidelines for safely using Java in most environments.
The end result is that push-based programs are powerful and flexible Web tools, and where the Web is directed. But these programs, by their nature, are inherently buggy and untrustworthy. Now take a look at the Java security model.

A QUICK INTRODUCTION TO THE JAVA PROGRAMMING LANGUAGE

The essence of Java is to be a portable and robust programming language for development of write-once programs. Java was created to alleviate the quandary of writing the same applications for numerous platforms that many large organizations faced in developing applications for large heterogeneous networks. To achieve this, the Java compiler generates class files, which have an architecturally neutral, binary intermediate format. Within the class file are Java bytecodes, which are implementations for each of the class’s methods, written in the instruction set of a virtual machine. The class file format has no dependencies on byte-ordering, pointer size, or the underlying operating system, which thereby allows it to be platform independent. The bytecodes are run via the runtime system, which is an emulator for the virtual machine’s instruction set. It is these same bytecodes that enable Java to be run on any platform. Finally, two significant advantages that increase Java’s security is that it is a well-defined and openly specified language.

While many systems subscribe to the security through obscurity model, Java achieves a significant level of security through being published. Anyone can download the complete set of Java source code to examine for themselves. In addition, numerous technical security groups and universities have done their own audits of Java security.

The second area where Java security is increased is through its architectural definitions. Java requires that all primitive types in the language are guaranteed to be a specific size and that all operations defined must be performed in a specified order. This ensures that two correct Java compilers will never give different results for execution of a program, as opposed to other programming languages in which the sizes of the primitive types are machine- and compiler-dependent, and the order of execution is undefined except in a few specific cases.

OVERVIEW OF THE JAVA SECURITY MODEL

The Java applet’s security model introduced with the 1.0 release of Java SDK considers any Java code running in a browser from a remote source to be untrusted. The model anticipates many potential attacks, such as producing Java code with a malicious compiler (one that ignores any protection boundaries), tampering with the code in transit, etc. The goal of the Java security model is to run an applet under a set of constraints (typically referred to as a sandbox) that ensures the following:
• No information on the user's machine, whether on a hard disk or stored in a network service, is accessible to the applet.
• The applet can only communicate with machines that are considered as trusted as itself. Typically, this is implemented by only allowing the applet to connect back to its source.
• The applet cannot permanently affect the system in any way, such as writing any information to the user's machine or erasing any information.

From a technical perspective, this sandbox is implemented by a layer of modules that operate at different levels.

Language Layer
The language layer operates at the lowest layer of the Java language model and has certain features that facilitate the implementation of the security model at the higher levels.

Memory Protection. Java code cannot write beyond array boundaries or otherwise corrupt memory.

Access Protection. Unlike C++, Java enforces language-level access controls such as private classes or methods.

Bytecode Verifier. When a Java applet is compiled, it is compiled all the way down to the platform-independent Java bytecode where the code is verified before it is allowed to run. The function of bytecode verification is to ensure that the applet operates according to the rules set down by Java and ensures that untrusted code is snared before it is able to be executed.

While the language restrictions are implemented by any legal Java compiler, there is still the possibility that a malicious entity would craft its own bytecode or use a compromised compiler. To deal with this possibility, Sun Microsystems architected the Java interpreter to run any applet bytecode against a verifier program that scans the bytecode for illegal sequences. Some of the checks performed by the verifier are done statically before the applet is started. However, because the applet can dynamically load more code as it is running, the verifier also implements some checks at runtime.

The bytecode verifier is the mechanism that ensures Java class files conform to the rules of the Java application. While not all files are subject to bytecode verification, those that are have their memory boundaries enforced by the bytecode verifier.
Security Manager. The function of Java security manager is to restrict the ways in which an applet uses the available interfaces and the bulk of Java's security resources are implemented via the security manager.

At the highest level, the security manager implements an additional set of checks. The security manager is the primary interface between the core Java API and the operating system and has the responsibility for allowing or denying access to the system resources it controls.

This security manager can be customized or subclassed, which allows it to refine or change the default security policy. Changing the security manager at runtime is disallowed because an applet could possibly discover a way to install its own bogus security manager. All of the Java class libraries that deal with the filesystem or the network call the security manager to ensure that accesses are controlled.

From a technology perspective, the security manager is a single interface module that performs the runtime checks on potentially dangerous methods that an applet could attempt to execute.

Security Package. The security package is the mechanism that allows for the authentication of signed Java classes. Those are the classes that are specified in the java.security package.

Signed applets were introduced in version 1.1 of the Java SDK and specifically are collections of class files and their supporting files that are signed with a digital signature.

The way in which a signed applet operates is that a software developer obtains a certificate from a certificate authority (CA) and uses that certificate to sign their applications. When an end user browses a Web page that the developer has signed, the browser informs the end user who signed the applet and allows the user to determine if he wants to run that applet.

Key Database. The key database works with the security manager to manage the keys used by the security manager to control access via digital signatures.

The Java Standard Applet Security Policy

The exact set of policies that are enforced by Java in a specific environment can be modified by creating a custom version of the security manager class. However, there is a standard policy that has been defined by Sun and is implemented by all Web browsers that implement Java applets. The standard policy basically states:

- An applet can only connect back to its source. This means, for example, that if the applet source is outside a company firewall, the applet is only allowed to talk to a machine that is also outside the firewall.
• An applet cannot query system properties because these properties could hold important information that could be used to compromise the system or invade the user's privacy.
• An applet cannot load native libraries because native code cannot be restricted by the Java security model.
• An applet cannot add classes to system packages because it might violate some access-control restrictions.
• An applet cannot listen on socket connections. What this means is that an applet can connect to a network service (on its source machine), but it cannot accept connections from other machines.
• An applet cannot start another program on the client workstation. This way, an applet cannot then spawn some other program or rogue process on the workstation. From a programming perspective, an applet is not allowed to manipulate threads outside its own thread group.
• An applet cannot read or write to any files on the user's machine.
• An applet can only add threads to its own thread group.

Java Language Security
This is not the place to detail the security features of the Java programming language, but a few of its most significant security-based features include the following.

Lack of Pointer Arithmetic. Java security is extended through lack of pointer arithmetic since Java programs do not use explicit pointers. Pointers are simply memory locations in applications. Consequently, no one can program (either maliciously or accidentally) a forged pointer to memory. The mishandling of pointers is probably one of the largest sources of bugs in most programming languages. To get around the lack of pointers, all references to methods and instance variables in the Java class file are via symbolic names.

Garbage Collection. Java garbage collection is the process in which Java deallocates memory that it no longer needs. Most languages such as C and C++ simply allocate and deallocate memory on the fly. The use of garbage collection requires Java to keep track of its memory usage and to ensure that all objects are properly referenced. When objects in memory are no longer needed, the memory they use is automatically freed by the garbage collector so that it can be used for other applets. The Java garbage collection engine is a multi-threaded application that runs in the background and complements the lack of memory pointers in that they prevent problems associated with bad pointers.

Compiler Checks. The Java compiler checks that all programming calls are legitimate.
E-Commerce and Java

Sun Microsystems has entered the E-commerce arena in a big way and envisions having Java at the forefront of E-commerce. To assist in that attempt, Sun has created a Java E-commerce architecture to promote that.

Components of the architecture are the Java Wallet, Commerce Client, Commerce API, and Commerce JavaBeans.

The Java Wallet is a family of products written in Java that enable secure electronic commerce operations. The Java Wallet combines the Java Commerce Client, Commerce JavaBeans components, the gateway security model, and the Java Commerce Messages to create a single platform for E-commerce. It should be noted that the components can be used independently of one another. The Java wallet is written in Java; thus, it can run in any Java-capable browser.

THREATS

McGraw & Felten describe in Java Security: Hostile Applets, Holes & Antidotes four classes of threats that Java is susceptible to:

• System modification: This is the most severe class of threats where an applet can significantly damage the system on which it runs. While this threat is the most severe, the defenses Java has to defend against it are extremely strong.
• Invasion of privacy: This is the type of attack where private information about host, file, or user is disclosed. Java defends against this type of attack rather well because it monitors file access and applets can only write back to the channel in which they were originally opened.
• Denial of service: Denial of service attacks are written to deny users legitimate access to system resources. Denial of service attacks take many forms, but are primarily applications or malicious applets that take more processes or memory allocation area than they should use, such as filling up a file system or allocating all of a system’s memory. Denial of service attacks are the most commonly encountered Java security concern and, unfortunately, Java has a weak defense against it.
• Antagonism: An antagonistic threat is one in which the applet simply annoys the user, such as via the playing of an unwanted sound file or displaying an undesired image. Many antagonistic attacks are simply programming errors. Most denial of service attacks can be classified as antagonistic threats, but the ones defined here are less annoying than their denial of service counterpart. Like their counterpart, Java has a weak defense against them.

USING JAVA SECURELY

By following some generic guidelines, and then customizing those guidelines for an environment’s unique needs, Java can be safely used in
most environments. Java security, like most computer security, is built on a lot of common sense. A few of the major issues are:

- Make sure that your browser is up to date: Many Java vulnerabilities have originated in browser design flaws. Staying with a relatively new release of a browser ensures that discovered security flaws have hopefully been ameliorated.
- Stay on top of security alerts: Keep track of advisories from CERT (www.cert.org), CIAC (www.ciac.llnl.gov), and the appropriate browser vendor.
- Think before you visit a Web site: If visiting www.whitehouse.gov, chances of downloading a hostile Java applet are much less than if visiting www.hackers.subterfuge.org. The bottom line, use your head when surfing the Web.
- Know your risks: Every company must assess its risks before it can really understand how to deal with the security risks involved with Java. If the risk of Java is too great (i.e., nuclear control centers), do not use Java; if the risks are more minimal (i.e., home), one can pretty much use Java with ease.

THIRD-PARTY SOFTWARE PROTECTION
There are numerous third-party software tools available to further secure Java and add protection against the potential security threats that Java can produce. Such products are a necessity for running push and active content applications.

- Finjan — SurfinGate & SurfinGate (www.finjan.com)
- eSafe Technologies — eSafe Protect (www.esafe.com)
- Digitivity — Cage (www.digitivity.com)
- Security7 — SafeGate (www.security7.com)

CONCLUSIONS ABOUT JAVA SECURITY
Java has an impressive security architecture and foundation, but one cannot rely on the sandbox model exclusively. Combined with poorly written PERL and CGI scripts, browser vulnerabilities, operating system holes, Web server holes, and more, there are plenty of potential openings in which a malicious or poorly written application could wreak havoc.

But by knowing what one’s risks are, combined with an understanding of what the Java vulnerabilities are, in association with active content protection, Java security is not an oxymoron.

Notes
1. An applet is defined as a Java program that is run from inside a Web browser. The HTML page loaded into the Web browser contains an <applet> tag, which tells the browser where to find the Java .class files. For example, the URL http://cnn.com/TECH/computing/JavaNews.html starts a Java applet in the browser windows since the source code contains the entry <applet code=Ticker.class>.
2. This article cannot list all of the details of the standard policy. For a thorough listing, view the Java SDK documentation set.

References

Ben Rothke, CISSP, is a New Jersey-based consultant with the Information Security Services group of Ernst & Young, L.L.P. He can be reached via e-mail at bennett.rothke@ey.com.