Java™ Security

J. Steven Fritzinger, Marianne Mueller
Sun Microsystems, Inc.

INTRODUCTION

Since its release in May of 1995, Java has swept across the Internet. With its promise of truly network oriented computing and a nearly universal system for distributing applications, Java is widely seen as the solution to many of the most persistent problems in client/server computing and on the World Wide Web. However, this same ability to distribute executables automatically over the network raises concerns about Java's effect on network security. These concerns have been heightened by the discovery of several security related bugs in existing Java implementations.

This paper discusses these concerns and how Java addresses them. It also describes several efforts underway to enhance and extend the Java security model. It is divided into three sections. The first section describes Java in general and discusses the security implications of Java. Readers who are already familiar with Java may wish to proceed to the second section which discusses computer security in general, how security affects networked systems and some misconceptions about security. Because these misconceptions are very common and affect how people approach new technology, readers who are unfamiliar with general security issues are encouraged to read this section carefully. The third section discusses Java security in particular, looks at how the security model is implemented, and describes upcoming extensions to the security model.

JAVA

The Java Platform

Java is a revolutionary new application platform from Sun Microsystems. Like other operating systems, the Java platform provides developers with I/O, networking, windows and graphics capabilities and other facilities needed to develop and run sophisticated applications. The Java platform also provides an important capability not found in traditional operating systems. This ability, called Write Once/Run Anywhere executables, allows Java programs written on one type of hardware or operating system to run unmodified on almost any other type of computer.

Applications written for traditional operating systems are tied directly to that platform and cannot be easily moved to another machine or operating system. This locks developers to that particular, often proprietary, system. If the application must be deployed on other platforms, the developers must port the application to those platforms. These porting efforts are often expensive and waste resources that could be used for new development. Because ports to the secondary platforms often lag behind the primary platform by several months, the application lock of traditional operating systems also forces the organization to support many different versions of the application. This administrative overhead makes networked computing with traditional PCs a very expensive proposition.[1]

With their Write Once/Run Anywhere capability, Java applications do not suffer from these problems. Developers working on a Sun Ultra computer running the Solaris operating system can produce an executable which also runs on Windows PCs, Macintosh and many other types of computers without any porting. This frees up development resources for other work and ensures that new applications and new versions of old applications are simultaneously available for all platforms in an organization.

The Virtual Machine

Java provides its Write Once/Run Anywhere capability through the Java Virtual Machine. The Virtual Machine is implemented on top of a machine's native operating system. Java applications run on top of the virtual machine. The virtual machine insulates the application from differences between underlying operating systems and hardware and ensures cross platform compatibility among all implementations of the Java platform (see fig. 1).

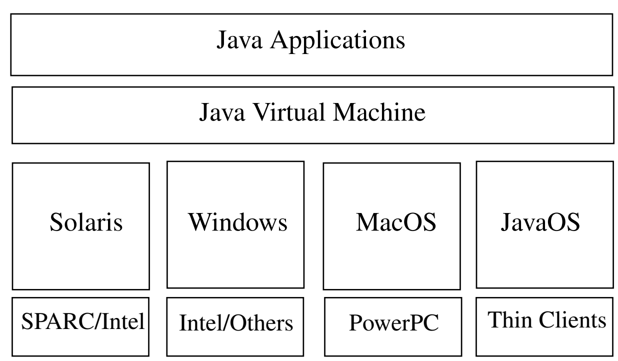

Fig. 1 The Java Virtual Machine sits between a native operating system and Java applications, allowing a single executable to run on many different systems.

The Java Virtual machine was first widely available in web browsers. Java-enabled browsers are currently available for

1 A recent report from Forrester Research estimates that, for companies aggressively managing their PC related costs, cost of ownership for the average PC ranges between $3,500 and $5,000 per year. Other studies have shown that for companies which are not closely watching PC related costs, cost of ownership can be as high as $12,000 per year.

the major versions of the Unix operating system, Windows 3.1, 95, and NT, the MacOS and OS/2 Warp. The Java Virtual Machine has also been licensed by every major operating systems vendor, including Apple, HP, IBM, Microsoft and SunSoft. These vendors will bundle the Java Virtual Machine with their operating systems. As these implementations become available over the next several months, Java will become a standard part of all important operating systems, and an expected part of every desktop.

Applets

Web Applets are one of the most exciting uses of the Java Platform. Applets are small pieces of executable code which may be included in Web pages and which run inside of the user's browser. While traditional web pages have been limited to simple text and graphics, applets allow web publishers to include sophisticated, interactive applications in their pages. For example, a stock broker might want to publish the results of a financial analysis model. With applets, instead of publishing a simple graph showing the results of the model, the broker could publish the model itself, along with connections to live stock market data and the customer's portfolio.

Security Implications

While applets solve many of the important problems in client/server and network-centric computing, they also raise new concerns about security. In traditional environments, companies could protect themselves by controlling physical and network access to their computers by establishing policies for what kinds of software can be used on their machines. These steps include building a firewall between the Internet and the company's intranet, obtaining software only from known and trusted sources, and using anti-virus programs to check all new software.

Use of applets potentially adds a new security vunerability. An employee searching an external Web site for information might inadvertently load and execute an applet without being aware that the site contains executable code. This automatic distribution of executables makes it very likely that software will be obtained from untrusted third parties. Since the applet is imported into the user's web browser and runs locally, this software could potentially steal or damage information stored in the user's machine on a network file server. Also, since this software is already behind the company's firewall, the applet could attack other unprotected machines on a corporate intranet. These attacks would not be stopped by traditional security measures.

Java protects its users from these dangers by placing strict limits on applets. Applets cannot read from or write to the local disk. Stand-alone windows created by applets are clearly labeled as being owned by untrusted software. These limits prevent malicious applets from stealing information, spreading viruses, or acting as Trojan horses. Applets are also prohibited from making network connections to other computers on the corporate intranet. This prevents malicious applets from exploiting security flaws that might exist behind the firewall or in the underlying operating system. While Java is the not first or only platform that claims to be a secure means of distributing executable code over the internet, it it perhaps the best known and most widely used.

WHAT IS SECURITY?

The Security Process

Effective security is an on-going process which must involve every member of an organization and touch every aspect of its operation. The strongest possible network and computer security does little to protect an organization which has not taken steps to ensure that its employees are trustworthy or to protect its physical assets from theft. Similarly, the best security mechanisms and procedures quickly fall into disrepair if they are not constantly reinforced by training and periodically updated to account for new threats.

Cost V. Security

Security is one means by which an organization can protect or extend a competitive advantage. By protecting valuable physical assets or proprietary intellectual property, security policies and procedures allow an organization to exploit those assets to the fullest. But there are costs associated with all security procedures and these costs must be weighed against the value of the assets protected by those measures and the potential harm which could be caused by the loss of that asset. A company which wished to advertise on the Web may be satisfied with a simple firewall to discourage electronic vandals. A large financial institute with billions of dollars at stake could justify much more elaborate security measures, possibly including public key encryption, dedicated, private networks and regular security audits. In extreme cases, public safety and national security may be at risk. For applications such as air traffic control and military and intelligence systems, the risks of connecting these systems to the Internet may so far out-weigh the benefits of increased communication that the most sensitive of these systems should never be connected (see fig. 2).

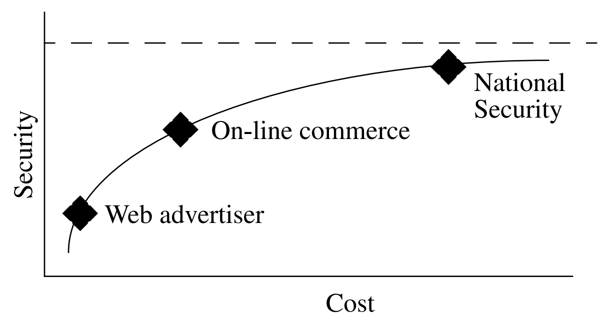

Fig. 2 Increasing security increases costs. Organizations must choose the appropriate trade off.

New Technology

Since no security system can ever be 100% secure, it is not meaningful to ask if a new technology or system is "secure". Instead one should evaluate the new technology in light of the existing cost/security tradeoffs. If the new technology makes it easier or cheaper to obtain the same level of security, that technology would be very attractive. If, on the other hand, the new system opens new security vulnerabilities and makes it more costly to achieve an acceptable level of security, the organization must carefully weigh the benefits offered by the technology and ask itself if these benefits are worth either the added risk they bring or the added expense required to manage these risks.

Usability

When calculating security costs, usability is an important, and often hidden, factor. If security mechanisms are too time-consuming or difficult to use, they can decrease productivity by taking time and resources which should have been directed to the organization's mission. Overly stringent procedures can actually weaken security. Users who find the policies difficult to follow may ignore the policies or implement them haphazardly. In extreme cases, where the policies are seen as bureaucratic roadblocks, users may actively sabotage the policies in order to "get the job done" (see fig. 3).

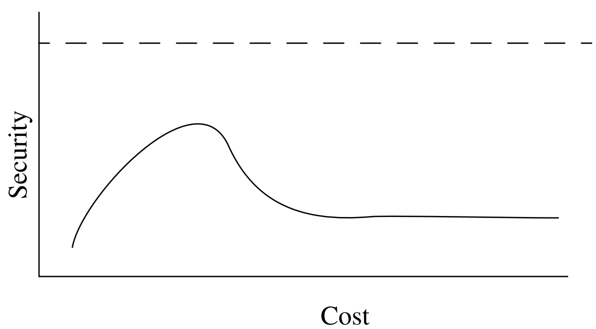

Fig. 3 Overly complicated and difficult to follow procedures reduce overall security and increase cost.

In general, it is very difficult to design easy-to-use or automatic security mechanisms which still effectively protect an organization's assets. Despite these difficulties, Java is able to provide transparent security mechanisms, which do not require any knowledge or action on the part of the end user. This is possible because Java's security model is intended to protect the end-user from hostile executables accidentally imported from untrusted sources. Limiting these so called "Trojan horses" is a much easier task than providing general network and physical security. Since Java's security model is intended to augment, not replace, these traditional security mechanisms, Java can provide a simple, usable solution to this simpler, more manageable problem.

Common Security Fallacies

Risk Avoidance

The most common security fallacy is that the goal of security is to eliminate all risk and vulnerabilities from a system. As discussed above, this is an unobtainable goal and little good comes from pursuing it. A company with a "zero tolerance" approach to security risks would be forced to disconnect itself completely from the Internet and thus would not benefit from the vast resources and near-universal connectivity it provides. Such a company would still be at risk from undetected viruses in commercial software, disgruntled employees and industrial espionage.

While this company spends vast sums of money and resources chasing the chimera of total security, its competitors with more realistic security policies would be concentrating on more practical matters such as exploiting new, "risky" technologies to better their competitive position.

Piecemeal Security

The risk avoidance fallacy is very common among computer users and managers. Fortunately, most security professionals recognize that their goal is risk management, not risk avoidance, and do not fall into this trap. Among these professionals, piecemeal security is a more common problem.

Piecemeal security is the tendency to look at small pieces of a system or network in isolation from the system as a whole. Because computer systems and especially computer networks can be extremely complex, it is of little value to examine individual aspects of the system. Informed security decisions can only be made by examining the entire system and looking for the unanticipated side-effects of adding a new type of software or network resource.

Piecemeal security often is the result of having several departments responsible for different aspects of security. If these departments do not work closely together, each can set policies without regard for how those policies affect security as a whole. This can create vulnerabilities at the borders between two departments and decrease the total security of the organization. These gaps are particularly dangerous since attackers may actively seek out areas in which several departments share security responsibilities or in which there is a gap between departments.

Steel Doors And Grass Huts

Piecemeal security can lead an organization to over-react to a perceived vulnerability. This is often the case when dealing with new technologies. A flaw found in the new technology prompts the organization to expend great effort patching the vulnerability, without first checking to see if this same vulnerability exists, undetected, in existing systems. Like steel doors on a grass hut, these patches, produced at great expense, close one possible hole but do little to increase the security of the system as a whole.

While the desire to build steel doors to protect against newly perceived threats can waste resources and slow the adoption of new technology, previously constructed steel doors can blind an organization to new or previously unnoticed threats. If the new found threat is not well-understood and is similar to the threat which motivated the construction of the steel door, false confidence in the elaborately constructed door's ability to protect against the new threat can slow the adoption of more effective measures.

Keeping Current

One of the most important parts of the security process is staying informed. New vulnerabilities in computer and network systems, and new attacks which exploit those vulnerabilities, are found regularly. Because of these new attacks, even the most secure installation will quickly become vulnerable if its security is not actively maintained by a well-informed, up-to-date staff.

The CERT[2] Coordination Center (CERT/CC) maintains an excellent set of on-line resources for security professionals. The CERT/CC evolved from an Advanced Research Projects Agency (ARPA) computer emergency response team formed in 1988 following the Morris Internet Worm. The CERT/CC collects and investigates reports of security attacks and new found vulnerabilities. They distribute this information as CERT Advisories, which document the vulnerabilities, list confirmed and rumored occurrences of attacks exploiting the vulnerabilities, and document patches and procedures to close the vulnerabilities.

Over the last several years the CERT/CC has documented approximately 10 to 20 new-found vulnerabilities and attacks each year. These vulnerabilities cover all aspects of computer security on systems ranging from mainframes to Microsoft Windows. CERT Advisories and other information can be found on their web site at `http://www.cert.org`.

JAVA SECURITY

The Sandbox

Java's security allows a user to import and run applets from the Web or an intranet without undue risk to the user's machine. The applet's actions are restricted to its "sandbox", an area of the web browser dedicated to that applet. The applet may do anything it wants within its sandbox, but cannot read or alter any data outside of its sandbox. The sandbox model is to run untrusted code in a trusted environment so that if a user accidentally imports a hostile applet, that applet cannot damage the local machine.

This approach is much different from that used in traditional operating systems. Because most operating systems allow applications broad access to the machine, especially in PCs where very little protection is provided by the operating system, the runtime environment cannot be trusted. To compensate for this lack, security policies often require a level of trust to be established in the application before it is executed. For example, an organization might require that before an employee runs an application obtained from the web, that application must be checked for viruses and its source code examined for malicious code.

There are two problems with this approach. First, the checks required to build trust in the application may be too complex and time-consuming to be practical. Few employees will take the time to read the source code of an application and compile it locally to ensure that it takes no hidden hostile actions. Second, virus checkers require constant maintenance in order to remain effective. They must be updated with samples of newly discovered viruses and must be installed on each machine. Also, many virus checkers can be turned off, either accidentally, as part of an installation procedure, or to save time when handling "safe" diskettes. If the checker is accidentally left off, the machine and possibly the entire organization are at risk.

Java solves these problems, and the usability problem discussed above, by automatically confining applets to the sandbox. End-users do not have to take any special action in order to ensure the safety of the machine. Because the sandbox prevents the actions required to spread a virus or steal information, instead of trying to identify a virus-infected executable or potential attacker, the sandbox does not require periodic updates with new viruses.

Applets And Applications

Java programs can exist in two forms: as applets, which travel across the Internet or intranet as part of a web page and run inside of the end-user's browser; or as traditional stand-alone applications. Only applets are subject to the security restrictions described above.

Java applications are purchased and installed just like traditional commercial applications. They may be purchased in "shrink-wrapped" boxes or imported over a network, and may be installed by users or system administrators (according to standard practice within an organization.) Since applications are not imported from outside the organization, and are (in theory) only installed by trusted individuals, Java applications add no new security concerns. Security comes from maintaining physical control over the systems, preventing end-users from downloading untrusted applications from the net, using virus checkers and other traditional security measures.

Building The Sandbox

The sandbox is made up of several different systems operating together. These systems range from security managers running inside of the application which imported the applet, to safety features built into the Java language and the virtual machine.

Class Loader

When an applet is to be imported from the network, the web browser calls the applet class loader. The class loader is the first link in the security chain. In addition to fetching an applet's executable code from the network, the class loader enforces the name space hierarchy. A name space controls what other portions of the Java Virtual Machine an applet can access. By maintaining a separate name space for trusted code which was loaded from the local disk, the class loader prevents untrusted applets from gaining access to more privileged, trusted parts of the system.

Applets downloaded from the net cannot create their own class loaders. Downloaded applets are also prevented from invoking methods in the system's class loader.

Verifier

2 CERT is a service mark of Carnegie Mellon University

Before running a newly imported applet, the class loader invokes the verifier. The verifier checks that the applet conforms to the Java language specification and that there are no violations of the Java language rules or name space restrictions. The verifier also checks for common violations of memory management, like stack underflows or overflows, and illegal data type casts, which could allow a hostile applet to corrupt part of the security mechanism or to replace part of the system with its own code.

Security Manager

The security manager enforces the boundaries around the sandbox. Whenever an applet tries to perform an action which could corrupt the local machine or access information, the Java Virtual Machine first asks the security manager if this action can be performed safely. If the security manager approves the action — for example, a trusted applet from the local disk may be trying to read the disk, or an imported untrusted applet may be trying to connect back to its home server — the virtual machine will then perform the action. Otherwise, the virtual machine raises a security exception and writes an error to the Java console.

The security manager will not allow an untrusted applet to read or write to a file, delete a file, get any information about a file, execute operating system commands or native code, load a library, or establish a network connection to any machine other than the applet's home server. This list is not exhaustive but does give a representative sample of the restrictions place on applets.

An application or a web browser can only have one security manager. This assures that all access checks are made by a single security manager enforcing a single security policy. The security manager is loaded at start-up and cannot be extended, overridden or replaced. For obvious reasons, applets can not create their own security managers.

Language Features

Java has several language features which protect the integrity of the security system and which prevent several common attacks. For example, Java programs are not allowed to define their own memory pointers or to access physical memory directly. This prevents an applet from accessing and modifying critical parts of the security system. The language tracks the type of newly created classes and objects so that an applet cannot forge its own class loader or security manager. The Java language also has several other checks for memory and pointer abuse which could weaken the security system.

In addition to making Java a more secure system, these language features also make Java programs safer and more reliable. Studies have shown that 40% to 50% of all bugs are caused by errors in memory management. By automating memory management, Java eliminates a large class of bugs; this results in more stable and reliable code.

Security Through Openness

In the past, many computer and network systems tried to maintain security by hiding the inner works and policies of the system. This practice, known as security through obscurity, assumed that if the system was presented as a black box then no one would expend the effort needed to discover the hidden vulnerabilities. The existence of the CERT/CC and a number of well publicized network attacks in recent years demonstrate that this assumption is unfounded; the box is never black enough. This is especially true for commercially successful systems. For such widely used systems, too many people know the internal workings of the system for the details to remain secret and the rewards for breaking into the system are too great.

Sun took the opposite approach, and published all the details of Java security model when Java was first released. This included the design specifications for the language mechanisms and the sandbox, and a full source implementation. This approach, dubbed security through openness, was intended to encourage security researchers to examine the Java model and to report any security flaws found; the flaws could be fixed before attacks based on those flaws could become endemic on the Web. Security through openness also allows any organization to study the Java security model in detail and make an informed assessment of the potential risks versus the benefits of the Java platform.

The Java Security FAQ

Keeping current is as important for Java security as it is for general security. To facilitate this, Sun maintains a Java Security Frequently Asked Questions (FAQ) page on the Java web site. This FAQ can be found at `http://java.sun.com/sfaq`. The FAQ contains more details on known vulnerabilities, the status of these vulnerabilities and, when available, dates and release numbers of the version of Java in which the vulnerabilities were fixed. More security related information can be found at `http://java.sun.com/security`.

Several other organizations are also tracking Java security. The CERT/CC has released several advisories on Java Security. These vulnerabilities have closely paralleled the vulnerabilities listed above and in the Java Security FAQ. Details are from the CERT/CC web site. Several other organizations, including researchers at Princeton University, have been investigating Java security. The Princeton findings can be found at `http://www.cs.princeton.edu/sip/`.

EXTENDING JAVA SECURITY

Security Modeling

While many experts agree that the Java Security model is basically sound, there is a concern that the model has not been examined in enough detail to ensure that the sandbox is as secure as is claimed. There is also the possibility that a particular implementation of the Java security model suffers from bugs and other coding errors which could be exploited by a malicious applet which wished to break out of the sandbox. Finally, there could be unexpected interactions between Java applets and other parts of the network which could be exploited. Problems which exploit all three of these categories have been reported.

For these reasons, Sun has initiated an independent, third party security modeling effort. The first step, being conducted by security consultant Blackwatch Inc. (`http://www.blackwatch.com`), will produce a Security Reference Model. The Reference Model will document Java's security model in rigorous detail.

The second step will be to develop a more rigorous security compatibility test suite based on the Reference Model. Since each Java licensee is free to re-implement portions of the Java Virtual Machine, the new test suite will ensure that both Sun and all licensees have correctly implemented the Reference Model. This test suite will be an enhancement to the test suite already used to ensure that Java implementations comply with the Java standard.

The third step will be to commission independent, third party assessments of Sun's reference implementation of the Java standard. This assessment effort relies on having a formal model specified so that the implementation can be assessed in the context of the assertions of the security model.

This review is expected to be complete by the fall of 1996.

New Security Facilities

The sandbox model described above protects the end-user's machine and networked computing resources from damage or theft by a malicious applet. Users can run untrusted code obtained from the network without undue risk to their system.

The sandbox model does not address several other security and privacy issues. Authentication is needed, to guarantee that an applet comes from the place it claims to have come from. Digitally signed and authenticated applets can be promoted to the status of trusted applets, and then allowed to run with fewer security restrictions. Encryption can ensure the privacy of data passed between an applet client and a server on the Internet. Work is underway to extend Java's security model into each of these areas.

Signed JAR files

All networked systems are potentially vulnerable to the "Man-in-the-Middle" attack. In this attack, a client contacts a legitimate server on the network and requests some action. The attacker, or man in the middle, notices this request and waits for the server to respond. The attacker then intercepts the response and supplies a bogus reply to the client. The client then acts on the bogus information, or possibly runs the program supplied by the attacker, giving the attacker access to the client machine. For example, an attacker might watch an Internet-based banking site. As clients visit the page which provides bill paying services, the attacker diverts the bank's responses and provides a malicious applet which mimics the bank's service, but also steals a copy of the user's credit card and bank account numbers.

This attack can be thwarted by applying "digital shrinkwrap" to the applet. We trust physical software we have purchased because its packaging shows who produced the software, and the shrinkwrap shows that the product has not been tampered with. If the producer has a good reputation for providing software which does not take any hostile actions against the user, then we can install the product with some degree of confidence.

"Signed applets" give us the same level of confidence in network distributed software. To sign an applet, the producer first bundles all the Java code and related files into a single file called a Java Archive, or JAR. The producer then creates a string called a digital signature based on the contents of the JAR. The full details of digital signatures are beyond the scope of this white paper. More details can be found in "Applied Cryptography," by Bruce Schneier, as well as numerous other cryptographic reference books.

JAR files solve another problem. Currently, many Java applets take a very long time to download and begin running. This can be annoying even for those users with a very high speed link to the Internet. The problem is that current Internet protocols move web pages across the Internet one file at a time. Since there is some overhead associated with each request for a file, web pages and Java applets which are composed of many small files might spend more time requesting those files and waiting for replies than they spend actually moving the information. Since a JAR file bundles all the information needed by the applet and its web page into a single file, the entire page can be downloaded with a single request. For many pages this will greatly reduce download times.

JARs and digital signatures can also be used for Java applications. While Java applications are more trustworthy than applets because they do not travel over the Internet and are subject to an organizations traditional security policies, applications are subject to several types of attack. For example, viruses spread by modifying existing applications to include a copy of the virus. Since a virus would not be able to produce a valid signature for the altered program, the Java system could detect that a signed application has been tampered with, and refuse to run it. Since the JAR signature system will work with all types of information, not just Java files, JAR signatures can also be used to protect data files and other information.

Signed JAR files will be included in Java release 1.1 and should be available by the end of 1996.

Flexible Policies

Since digital signatures allow us to assign to Java applets the same level of trust which we assign to shrinkwrapped applications, it may be useful to relax the Java security restrictions for some applets. For example, it would be useful if the home banking applet described above could establish its own directory on the user's hard disk. It could store account and credit card numbers, passwords, PINs, and other frequently used information so the end-user would not have to constantly re-enter that information.

Signed applets can be used to create this environment. If the end-user has previously told the Java system that a particular web publisher, say a bank or credit card company, is trusted and a signed applet from that publisher has arrived over the Internet and been authenticated, then the Java Security

Manager could allow that applet out of the sandbox, and treat it as an application.

The Security Manger could also enforce different levels of control based on how much a particular publisher is trusted, or on how much the Internet as a whole is trusted. For example, a very security-conscious user could configure the system to allow signed applets to run only inside the sandbox, and to prevent any unsigned applet from running at all. Another user might configure the system to allow the banking applet to access only one particular directory on the hard disk, while a net gaming applet could access another directory and all other applets are restricted to the sandbox.

Auditing

Auditing is another important security feature. Auditing software maintains a record of everything which happens on the system. When something goes wrong, either through an accident or a bug, or because of an attack, the audit trail allows systems administrators and security personnel to figure out what happened, and how to prevent a reoccurrence in the future. While auditing cannot prevent accidents and attacks, once things go wrong, it is an important tool for cleaning up the mess.

While some versions of the Java platform include limited auditing features, there is no standard set of auditing capabilities on which an administrator can rely, and those features that do exist do not record as much detail as is often needed. Efforts are under way to define what standard features need to be included in every Java implementation and how these features should be implemented.

Encryption

While the sandbox and signed applets can protect against hostile applets and man-in-the-middle attacks, information traveling between the applet and a server on the Internet is still vulnerable to theft. This is because the Internet itself is an insecure medium. An attacker attached to a central portion of the Internet can read all information which travels through that portion of the Internet. The attacker could listen to all traffic bound for a major bank or mail order company, and simply read credit card numbers and other information off the wire as it passed. To secure against this type of attack, all information flowing between the applet and its server must be rendered unreadable by encrypting it.

Several Java encryption facilities are being developed. These facilities will allow applet developers to select the type of encryption algorithm used, to negotiate with the server to create the keys used in the encryption and to do the actual encryption of the data.

While there are few technical challenges to implementing the cryptographic functionality, the US government imposes strict export controls on encryption technology. Since Java is available world-wide, any proposed cryptographic system must comply with these laws. Ensuring this compliance may delay the release of the facilities.

SUMMARY

The Java platform supports Write Once/Run Anywhere applications. This, combined with the easy distribution mechanisms provided by the World Wide Web and Web-like systems called intranets, makes Java a powerful tool for many network based systems. The mobile applications which Java enables — applications that automatically migrate over the network to where they are needed —solve many persistent problems in application distribution and systems management.

While mobile applications solve the software distribution problem, they also make it more likely that end-users will unintentionally import hostile applications into the corporate network. Java addresses these concerns by running all untrusted applications in a protected area known as the sandbox. Applications running in the sandbox can only access local and network resources through a limited set of trusted mechanisms. The sandbox model gives users the advantages of easy, ad-hoc application distribution while it protects them from potentially malicious applications.

Several efforts are underway to further enhance the sandbox model. Independent contractors are reviewing the design of the sandbox to ensure that it provides adequate protection. Future releases of Java will provide applet signing, which acts as digital shrinkwrap. Support for flexible security policies, encryption and other more advanced security features are also being added.

Any organization which is considering adding Java applications or Java enabled software to its network should carefully consider how Java will affect their current security policies. While no set of security policies can ever eliminate all risk from a networked environment, understanding how Java's security model works and what sorts of attacks might be committed against it, keeping current with new developments by both attackers and other security officers, and evaluating Java in light of the organization's overall security policy can reduce risks to an acceptable level.