Java Applets and Security

Petri Niinimaki & Panu Markkanen
Nokia Telecommunications
Teknologiantie 3C, FIN-90570 Oulu,
Finland
Petri.Niinimaki@ntc.nokia.com
Panu.Markkanen@ntc.nokia.com

Jorma Kajava
University of Oulu
Linnanmaa, FIN-90570 Oulu, Finland
kajava@rieska.oulu.fi

Abstract

The paper summarizes the collected information about Java applet security which has become an object of interest for several parties. The security is based on the sandbox concept where the executable content is located and run isolated from the rest of the system. In this paper the key security issues related to the Java applets are discussed. What the problems are, how they are solved, what holes are still left, and what does the future bring along to this fast growing business related to networks.

The used references are books and articles found from the WWW. The Web offered much information which probably was correct but one could not be sure about it. Therefore a critical approach was used with the information gathered from the Web.

Keywords: computer and information security, Java applet security, WWW security.

1. Introduction

Java is a several years old programming language developed by Sun Microsystems. It has quickly become a serious way of implementing platform-independent software. Java is nowadays used to create web applications. The power - and the danger - of Java is that Java's executable contents, applets, can be loaded over the network and executed by the web browsers in the user's computer.

Since one of the initial goals of Java was for it to be used in embedded systems with minimum resources, the language is designed to be compact and also to use small amount of hardware resources. There are special Java processors being built to optimize the code execution.

With the immense size of today's applications the platform independence is becoming the prime factor for software development. Java provides platform independence via Virtual Machine (VM) implementations on multiple platforms.

The platform independence also provides flexibility into client/server architecture by enabling cost-effective selection of clients. Clients running the Java VM are granted access to the server. Then the actual client software is downloaded and executed. This enables the server to require the use of custom client software because the client does not have to acquire the software beforehand. Also the distributed computing is a practical reality now [1]. Distributed development architectures allow applications to be divided into pieces, each of which can exist in different locations. The ease of access increases security concerns.

When the applications are written in Java the management of software resources can be centralized. Since all the software resides in the central archive, few maintenance operations are needed in the network terminals. This brings down the maintenance costs drastically.

Many companies have been rushing into the Web just to keep up with the latest development with little regard on the impact it has on their information security. The security aspects of web applications have been brought to the front along with the Java applets which have become very popular among the web page builders.

Java applets are offering power and expressiveness especially to the World Wide Web (WWW or Web). The Web is traditionally based on scripts run on servers which is quite limiting. Applets give the possibility to run real programs locally. [2]

The security of web applications is essential for the organizations which have internet access. E.g. electronic commerce needs reliable web applications with secure transactions. The users should be able to trust that the web applications cannot break the integrity of the information stored on their computers.

Java is under constant reviews and has not yet been standardized. It is a relatively new programming language and therefore its security features are not verified in real environments. As Java applets are loaded from the net to the user's computer and executed without prior notice malicious code can try to penetrate the security defences. This brings a new aspect to computer and information security and has received much attention throughout the information processing community.

This paper has been written to offer a view to Java applet security, the problems, solutions and future expectations. The main information source for this paper was [3]. Additional information was obtained from WWW articles using a critical approach. There is a lot of information available in the Web but its reliability is somewhat uncertain.

2. Java applet security

Java programs can be divided into two different types: applications and applets. The applications are like any other program, just made with Java. The applets are executable contents which are meant to be run in the context of a web browser. At the moment, the applets are the most important area of Java application.

According to Sun Microsystems Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language [4]. Java includes e.g. the following features:

- object-oriented, data-centric model
- native multi-threading support
- restricted access to runtime environment
- runtime verification of code integrity
- garbage collection
- no pointers
- no macro support
- compiler-generated symbolic byte code (interpretable)
- digital signature
- encryption

Data-centric model means that the data objects possess associated methods which perform actions on data objects.

The removal of pointers removes a significant error source. For example in C/C++ programs the most errors are related to pointers. Deletion of used, unneeded resources is automated in Java which also removes a common problem encountered in other programming languages.

Digital signature and encryption are added to Java just recently (version 1.1) to make up the security features. As they are quite new additions, there are most likely some holes in them. Some have been found but others are still waiting to be detected.

The digital signature can be applied to files using a special tool. The signed files are called Java Archive (JAR) files and they can contain Java classes and other data, such as images and sounds. Applets in JAR files are loaded by the appletviewer and in case of trusted signature the applet has full rights like a local application.

There are several weaknesses in Java according to [5]. For example the Java DataBase Connect for database connectivity requires a direct connection to the database to work. This is not acceptable in information security. Also, the WWW browsers[1] set so many restrictions that applets cannot be used in critical applications. [5]

The security of Java applets is critical since they are exceptionally easy to download, sometimes even without the user noticing. Restricted intranets face the threat of users downloading and executing harmful applets while netsurfing, thus exposing the intranet to outside influence.

In the following sections the three-tiered Java security foundation is discussed. Applets are restricted to a 'sandbox' which relies heavily on the security foundations. The sandbox offers an area in the web browser where the applet is located and run without the fear of the applet altering the data outside this area. [6]

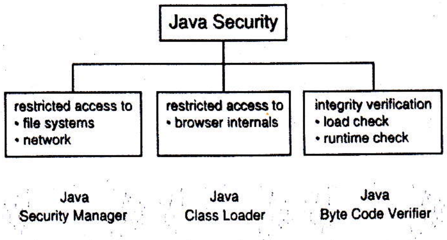
Java Security
restricted access to • file systems • network
restricted access to • browser internals
integrity verification • load check • runtime check
Java Security Manager
Java Class Loader
Java Byte Code Verifier

Figure 1. Security foundations in Java

[1] Actually, the restrictions are set by Java security model and not by WWW-browsers as [5] states.

2.1 Byte Code Verifier

In the case of applets the Byte Code Verifier is located in the web browser. It is responsible for verifying the Java byte code before the code is allowed to be run. The Verifier ensures that the code which may or may not have been created by a Java compiler is in accordance with the rules [3]. The key assumption made is that the byte code has come from an untrusted source.

The work done by the Verifier is the most complex part of bytecode verification process. The four passes of verification include: ensure the class file has the right format, verify other details without looking at the bytecodes, verify the bytecodes of every class (done by the Byte Code Verifier), and verify the classes at load-time. [7]

The Verifier reconstructs the type state information with the help of symbolic information included in the code.

- stacks are not overflowed nor underflowed
- correct parameter types
- no illegal conversions
- no access to restricted interfaces
- valid register accesses and stores
- correct format

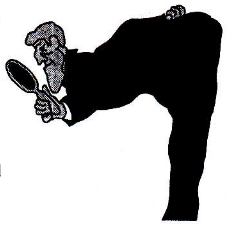

Figure 2. Java Byte Code Verifier

The Verifier removes much of the checking that the interpreter should otherwise do. Therefore the interpreter will run faster. Even though the Verifier acts as the gatekeeper, the Java run time system still has to be faultless. As yet, Sun Microsystems has had less flaws in its run time system than the other vendors (Netscape, Microsoft, Borland, Symantec).

2.2 Class Loader

The Java Applet Class Loader determines when and how an applet can add classes to a running Java environment. Class Loader is used when an attempt is made to load a class from any other place than the local machine. It makes sure that the important parts of Java run time environment are not replaced by classes loaded from outside the computer. The Class Loader utilizes name spaces to which the classes are separated according to their origin. [3]

A running Java environment can have many Class Loaders active, each defining its own name space. Each Loader will load every applet into its own name space so the applet does not see the other possible applets present. The applet will see only its own classes and all of the classes in the standard Java library API. [3]

In case a class tries to reference another, the Class Loader will perform a search in this particular order:

1. search local name space where built-in classes are
2. search name space of the class making the reference
3. search explicitly defined external name space with public methods

Applets downloaded from the Web cannot create their own Class Loaders nor invoke methods in the system's Class Loader. This ensures that the loaded classes stay in their sandbox and cannot breach out of it.

2.3 Security Manager

Java Security Manager restricts the ways an applet uses visible interfaces. Code in the Java library consults the Security Manager whenever a potentially dangerous operation is attempted. The Security Manager has the rights to generate a Security Exception denying the operation. [3]

The Java run time library is written so that all access requests are referred to the Security Manager. The main duties for Security Manager are to:

- prevent installation of new Class Loaders
- protect threads from each other
- control the creation of OS programs
- control access to OS processes
- control file system and socket operations

- control access to Java packages

The Security Manager is fully customizable though not by applets. The user can customize the level of security for each module by using a special configuration file. The Security Manager prevents the Java applets from:

- reading or writing files on client system
- deleting or renaming files on client system
- creating a directory on client system
- listing the contents of a directory
- checking whether a file exists
- obtaining information about a file
- creating network connections to hosts other than the one the applet came from
- listening for or accepting network connections from any port on the client system
- creating a top-level window without 'untrusted window' banner
- obtaining user information
- defining any system properties
- running any program using Runtime.exec()
- making the Java interpreter exit
- loading dynamic libraries on client system
- creating or manipulating other threads
- creating Class Loader or Security Manager
- specifying any network control function
- defining packages that are part of packages on client system

2.4 Summary

The sandbox is of utmost importance to Java applet security. The sandbox, which is mainly based on Byte Code Verifier, Class Loader, and Security Manager, allows untrusted applets to be executed in a trusted environment without fear of corruption or subterfuge [1]. Also the Java VM enforces the sandbox functionality by its own security measures.

Java relies on a practical approach to security. The security measures build on preventing predicted attack scenarios. The problem with this approach is that it takes into consideration only the known attacks. With the theoretical approach the security could be formally proven in all situations.

3. Attack scenarios

The threats formed to the system by Java applets can be divided into four basic categories (listed in order of decreasing severity) [3]:

ATTACK SCENARIOS

- ➢ system modification
- ➢ invasion of privacy
- ➢ denial of service
- ➢ antagonism

Figure 3. Java applet attack scenarios

The system modification is the most serious threat and that is why Java developers have put a lot of effort into preventing it. The modification attack includes file altering and deletion, memory alterations and process / thread killing. [3]

Invasion of privacy involves disclosing information about a user or host machine that should not be published. Confidential information can be obtained in numerous ways: From e-mails, microphones, process tables, and file access. The file I/O is heavily guarded in Java but also there is a weakness related to applets: They always have a channel open back to their original host server. As a result, it is very easy to send information once the applets get hold of it. Mail forging is a serious threat where the applet loaded from a computer running in a second computer, will forge a message to a third computer. [3]

In the denial of service attack the system resources are reserved by the attacker. This could happen by filling up the file system, using all file pointers, allocating all of a system's memory or using all of the machine's CPU time. Though the denial of service is a real concern, the attack is not prevented in Java. This can form a problem as the denial of service attacks are very easy to implement and in addition the prevention is difficult. [3]

Antagonizing attacks are merely annoying to the user but still form a threat. Some attacks can be classified as denial of service or antagonism depending on the context they exist. Examples of antagonizing attacks could be playing unwanted sound files or showing text or images on the monitor. [3]

4. Analysis

The references used were chosen using a critical approach. Known reliable sources were used and in case of unknown source the information reliability was considered and possibly cross-checked from other sources.

As stated earlier in this paper Java is relatively new language and it has not been tested comprehensively. A formal security model for Java is obviously needed to provide a way to verify the security. Without it there will be constant surprises when the users find holes from the security just by coincidence or on purpose. The formalization process requires a complete specification of Java source semantics which is not currently available. Also the specification for byte code semantics is needed.

As the number of Java vendors increases, the number of Java security implementations also increases. This introduces threats, since the lack of formal security model prevents from verifying these new implementations. The main problem is how to ensure the security of the Java solutions shipped by non-Sun vendors without the formal model. There are flaws in the particular versions of popular browsers that have gained a CERT recommendation of disabling Java [8].

One way to eliminate the security risk related to Java applets being loaded over the net is to disable Java execution from the browser. This is a radical move, but it is the only way to be certain that there will be no Java applets loaded, whether they are trusted or not to the user's computer. If the user trusts some WWW-sites, he/she can enable the Java applets while surfing on those particular sites and keep them disabled when going outside those sites.

There is also a possibility to sign applets wiih the user's digital signature. Other people can register this user's signature into their browsers if they trust this user and have obtained the public key for the user. After that, the browser will allow applets with that digital signature to be executed. This offers a significant improvement to the applet security as companies can examine applets and after finding them harmless, sign them with an acceptance label.

Sun has added the human judgement factor into Java in form of digital signature. This could open a worm hole for the hostile applets to damage the computer system as humans are often the weak link when it comes to security.

Cryptography is being added to Java Developer Kit (JDK), but since U.S. federal government does not allow strong encryption tools to be included in software exported or used outside the U.S., this cryptography option will be quite insufficient.

Java still lacks auditing ability which certainly is one important security feature. The history information is invaluable when a break-in or system failure must be examined. Sun is currently working on this issue.

Authentication protocols are being studied at the moment. If some solid authentication mechanism could be added to Java it could open the Java applet technology to wide commercial use, even to be applied in firewalls. Access control lists are also coming into use, enabling further restrictions to systems and services.

The goal of Java security model is to ensure that untrusted applets cannot steal or damage information on a computer running a Java-enabled browser [9]. Since an implementation bug in a trusted applet could open a hole that could be exploited by an untrusted applet, the applet's design matters.

Thanks to Java Byte Code's clarity, the byte code is very easy to decompile. This leads to problems in the companies which want to use Java as a development language but still fear that their code will be decompiled and exploited by other companies. There is already an answer to this problem as obfuscators are being developed. Another problem is that crackers might decompile the target Java program to find out how it is made and what are its weaknesses.

Critical pieces of the Java environment directly affecting security include Java VM and the built-in libraries of JDK. If any of these pieces have errors, the entire security system will break, regardless of which vendor creates the other parts. From a security standpoint, the Web browser's implementation of the Security Manager is much more critical than the implementation of the Class Loaders as without Security Manager, a Java applet has the same rights as a local Java application [2].

The Security Manager has a relatively weak control on the top level window creation by forcing the window to be marked as unsafe. This creation could be controlled more tightly in order to prevent availability attacks which, for example, use the system's resources by creating windows. Also there could be more control over display and audio resources' access.

Java is at the moment the best choice to be used as a relatively safe means to provide extra value to the Web world. ActiveX technology from Microsoft lacks the security which is a very important feature when executable contents are used in the Web.

ActiveX relies on a fundamentally different approach than Java. The human judgement used in the ActiveX security (in form of digital signature) is always dangerous. Humans do not behave strictly according to rules which is needed to reach acceptable security levels. People might accept malicious ActiveX (as well as digitally signed Java applet) components by accident or on purpose.

So far there has been only few incidents where hostile Java / ActiveX programs have done some serious damage to computer systems. It seems that the skillful people have not moved into making malicious Java / ActiveX programs yet. [10]

5. Conclusion

Java applets have been, are, and will be insecure. The question is whether they are secure enough to be used in serious applications. Java applet is still a much better alternative than Microsoft's ActiveX, which does not include sufficient security mechanisms.

For systems with high security demands, the Java is not recommended because the trade-off between the security aspects and the power is not good enough. But in systems where security is not the key issue, Java offers benefits along with acceptable security risks.

The Java applet technology is already being used by software houses, e.g. Corel Office is published as a combination of Java application and applets. The popularity of Java is growing all the time. Therefore, rapid development of Java's security features and also the fast bug fixing are essential - a widely used technology must not have security holes.

In the future the network management could be done using Java applets. The Java applet security would have to be at a sufficient level before this could happen on a large scale.

The portable computer's security along with so-called dummy agents is a potential area of further studies. The dummy agents can search for certain information from the Web when the user gives an order. The concept of having a portable computer and a mobile phone embedded into the system brings a new security viewpoint. The security issues must be examined carefully especially when the agents run in the computer get smart and start working independently. They might even be able to use your money on their own.

References

[1] Anonymous (1997) Secure computing with Java: Now and the future. Sun Microsystems. http://java.sun.com/marketing/collateral/security.html

[2] Bank J.A. (1995) Java Security. http://swissnet.ai.mit.edu/~jbank/javapaper/javapaper.html

[3] McGraw G. & Felten E. (1996) Java security: hostile applets, holes & antidotes. Wiley Computer Publishing. http://www.wiley.com/compbooks/

[4] Anonymous (1995) The Java language: a white paper. Sun Microsystems. http://java.sun.com

[5] Lappalainen P. (1997) The Exploitation of Internet in Digital Mobile Phone Networks (in Finnish). Department of Information Processing Science, University of Oulu, 70 pg.

[6] Fritzinger J.S. & Mueller M. (1995) Java Security. http://www.javasoft.com/security/whitepaper.txt

[7] Yellin F. Low Level Security in Java. http://www.cs.wisc.edu/~lhl/cs740/java.html

[8] http://www.cert.org

[9] Leach B. (1995) The Java security mechanisms. http://compsoc.lat.oz.au//~leachbj/java/security.html

[10] http://www.cs.princeton.edu/sip/java-vs-activex.html