Proceedings of International Conference
on Cryptology and Information Security

# A Java Security Model Based on Information Flow Control*

Chih-Yuh Chang, Chin-Laung Lei and Wen-Shenq Juang

Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, R.O.C.

**Abstract**

*In this paper, we propose a new Java security model based on information flow control to enhance the system security. Moreover, we also simplify the management of the security of Java environments and reduce the chance of falling into traps caused by careless operations. Our proposed security model provides a more secure and user-friendly Java environment for naive network users and greatly reduces the load of network administrators.*

## 1 Introduction

Due to the continuing and fast progress of internet, network users expect more and more advanced network services. They quest for more powerful services than the traditional ones such as TELNET, FTP, GOPHER, etc. Hence, CERN proposed a new architecture, the World Wide Web (WWW), to provide network users a new solution. WWW not only combines almost all existing services but also provides a new distributed multimedia environment for Internet. Network users can use it to browse multimedia documents on line or off-line. Moreover, WWW could be extended by a standard interface, CGI (Common Gateway Interface), for programmers. Any one can code external functions with his favorite programming language to extend the capabilities of WWW. However, CGI has some inherent limitations within its architecture and is quite difficult for ordinary network users.

In 1995, Sun Microsystems announced the Java and the associated HotJava Web browser. Java has the potential to enhance the capability of WWW and more generally mobile code technology, and it augments the present WWW capabilities to transmit static information with new capabilities to transmit programs, called applets, in a portable environment. Since Java is designed to be transportable across networks and executed in the browser, security is extremely important. At present, access control list (ACL) is the common method to protect users' system resource in the Java environments.

Since all permitted operations of system resources can be recorded into ACL, alterable options of security consideration are quite complicated. Although system administrators can tune the details of ACL to fit their security policy completely, they would have a hard time in doing it. Typical network users will easily fall into some traps of evil hackers. ACL is not secure enough, too. It can be attacked by the Trojan Horse program. Hence, we propose a new security mechanism based on the information flow control to provide a simpler, more secure and more user-friendly Java environment.

The schemes of the information flow control are concerned with the flow of information from one class to another. The operations of system resources are permitted if the flow of information is allowed. There are three typical information flow control models, the BLP model, the Biba model, and the combined model. Each model has some of the desired security properties. Our scheme is based on the combined model, which combines the first two models and satisfies the properties of confidentiality and integrity. For a real Java environment, we design a set of classes for the possible types of applets, a set of classes for the four types of system resources, and a set of rules to control the flow of information.

Although information flow control is more secure and easier-to-manage than ACL, the flexibility of adding or deleting a permitted operation in ACL is lacking in information flow control. In addition, the security of information flow control would be too complex to analyze if there are too many classes. Hence, we use ACL to control some system resources, such as the file system, that are hard to control by using information flow control. Information flow control and ACL form two layers of protecting mechanisms in our proposed model.

* This research is supported in part by the National Science Council of the Republic of China under grant NCS-86-2221-E-002-014.

This paper is organized as follows. In section 2, previous work on information flow control and the current Java security environment are reviewed. Our new Java security model based on the lattice model is described in section 3. Then, we compare the security of our model with that of the current Java approach in section 4. Finally, concluding remarks are given in section 5.

## 2 Background

Java is a language developed at SunLabs [8]. The programming language and environment are designed to solve problems in modern programming practice. It originates as part of a research project to develop advanced software for a wide variety of networked devices and embedded systems. The goal is to develop a small, reliable, portable, distributed, and real-time operating environment. Since the Java language compiled code is designed to be transportable across networks, we must watch out for the same threats caused by copying programs to different computers. Three traditional threats are considered: Trojan horses, viruses, and denial-of-service attacks [5]. To prevent these threats, a Java system must provide security mechanisms to address various security concerns. In the current design, the security architecture of Java is separated into three layers to verify and monitor the instructions of Java program.

The first layer, compile-time checking, is bound up with fundamental features of Java. Its security mechanisms are derived from the Java language specification. The Java language and the compiler constitute the first line of Java security. They are precisely specified, and are not configurable. The second layer, bytecode checking, is a run-time checking. A trustworthy compiler ensures that Java source code does not violate the safety rules, but someone could alter the compiler to produce code that violates them. So, before executing any code fragment, the runtime system is subjected it to a series of tests. The third layer, access right checking, is also a run-time checking. The major difference between the second layer and the third layer is the configurability for users. There exists a security manager to monitor all access permission of system resource. For security concerns, the security manager denies any access request of system resource by default unless the user explicitly changes it. Since users can configure what applets can and cannot do, it would be a hard job for users to set the complicated configuration appropriately not to mention optimally. Thus, a good scheme for the security manager is necessary to loosen the security administration loading and enhance the system security.

There are two major approaches for controlling accesses to system resources: the access matrix model and the information flow control. The access matrix model is the fundamental scheme to control access, and information flow control can be considered as an extension of the access matrix model with more security.

In the access matrix model, the access rights of each user to each resource are defined as entries in a matrix. The most important feature of the access matrix approach is that accesses are controlled by classification of users and resources. Only the rights of users to access resource are checked. The basic disadvantage of the access matrix approach is its vulnerability to attack by Trojan Horse programs. Although the access matrix is simple, its direct implementation would be very inefficient and expensive since it is usually sparse. Thus, the following two variant methods are usually used in actually implementations: access control lists, and capabilities. Current Java security scheme uses access control lists.

On the other hand, the approaches of information flow control are concerned with the flow of information from one class to another. In a system, information actually flows from one object to another. An object can be informally defined as a container of information. The flow of information is usually controlled by assigning every object a security class, also called a security label. Information can flow from object X to object Y if and only if information can flow from security class of object X to security class of object Y.

The security for information flow control deals with three main goals: Confidentiality is to ensure that privacy information should not be accessed by anyone without such a right. Integrity is to ensure that information is not changed. Neither the system, which stores and transmits data, nor unauthorized users should be able to corrupt that data. Availability is to ensure that authorized users can make effective use of a computer system.

There are several some models of information flow control. The Lattice model defines the information flow policy and axioms by Denning [1, 2]. Bell and LaPadula represented the security kernel as a finite state machine, and the security rules define allowable transitions from one secure state to the next. They defined a model commonly bearing their name, and we call it BLP model in short. The BLP model satisfies the property of confidentiality by its security rules. Confidentiality considerations motivated the mandatory controls in the BLP model, while the Biba model proposed that similar controls could be formulated for integrity. The basic concept in the Biba model is that low-integrity information should not be allowed to flow

to high-integrity objects. In the usual formulation of the Biba model, high integrity is placed toward the top of the lattice of security classes, and low integrity at the bottom. With this formulation, the permitted flow of information is from top to bottom, directly opposite to that of the BLP model. It is often suggested that the BLP and Biba models could be combined. Thus, both confidentiality and integrity are considered. Unfortunately, none of the existing models can deal with the problems of availability. Our proposed scheme is based on the combined model to satisfy both confidentiality and integrity which current Java security scheme lacks.

## 3 The proposed Java security model

Since the current Java security mechanism uses access control lists, the configuration for system security is rather complicated for users. Moreover, typical network users are prone to fall into some traps of evil hackers. To provide a simpler, more secure and more user-friendly Java environment, our model is based on information flow control enhanced by access control lists. We classify users into three types: administrators, developers, and common users. A Java program will be assigned to a default subject depending on the type of users and the environment. All system resources, treated as objects, are divided into four major parts: the file system, the network connections, the system information, and the executable system programs. We also design rules for different types of system resources to deal with all potential security operations.

Whenever an applet requires an object that it cannot access, the user would be prompted by a dialog box popping up for deciding whether or not to allow the applet to alter to a higher security class. The users can disable this feature, in that case, an applet will not be able to access any object which it is not entitled to. Thus, with our approach, users can maintain the security of their systems easily and could refer to the suggestions and warnings displayed with pop-up dialogs. These suggestions or warnings could be generated by analyzing the difference between any two classes in our model for the system.

### 3.1 The set of subjects and objects

Users can be separated into three types according to the attribute of their work:

- Administrators, who maintain the system workable and secure, know how to manage system security and need the permissions to modify the system.
- Developers, who code applications, need the permissions to design and test their programs.
- Common users, who execute the programs for their goals, just need the permissions to run programs in the secure environment.

The set of subjects is used by users or Java programs including applets and standalone applications. The set is described below:

- SA: System Administrator. This subject is for the super users who manage the whole computer or for the trustworthy Java programs that perform the job of maintaining the system.
- JS: Java Standalone application. By default, Java standalone applications are assigned to this subject.
- JA: Java Administrator. This subject is for the managers who maintain the Java related files including executive programs, class library, configurations, or for the Java programs that automatically monitor and update the Java environment.
- DL: applet Developer that loads applets from Local file system. By default, the applets written by Java developers and loaded from local file system will be assigned to this subject.
- DN: applet Developer that loads applets from Network. The difference between DL and DN is that DL is for the applets loaded from local file system, and DN is for the applets loaded from untrustworthy network.
- BL: applet Browser that loads applets from Local file system. For common users, the applets are invoked from inside the web browser and are loaded from local file system.
- BN: applet Browser that loads applets from Network. If the applets are invoked from inside the web browser by common users and are loaded from untrustworthy network, they will be assigned to this subject.
- IA: Interactive Applet. When a trustworthy applet whose subject is BL or BN requires more network access right, users can upgrade the applet to this class.
- LA: Local Applet. When a trustworthy applet whose subject is BN requires some secret system information, users can upgrade the applet to this class.

We also partition all system resources into four major parts:

- File system: The typical permissions, read, write, and modify, can be easily monitored by all the model of information flow control. In our model, access to a file in the system proceeds as follows.

(a) If the file is created by a subject, it can only be accessed by the subjects according to the original rules of the combined model.
(b) If the file is not created, it can only be accessed by the subjects according to the access control list.

- Network connection: Since sensitive information may be leaked out to a remote host, the permission to connect with other networks can be controlled by the rules of information flow control.
- System information: Some system information may be useful for breaking into the system and should be kept secret by the rules of information flow control.
- Executable system programs: The permission to execute the system programs is controlled by the rules of information flow control, and the details are recorded into the access control list.

The set of objects is described as follows:

- Files that are created by a subject. Such a file is assigned a class and permitted to access by the rules of the combined model. This object is hidden with each subject.
- FR: File system Read ACL. The access control list maintains the information about what files and directories can be read by subjects.
- FW: File system Write ACL. The access control list maintains the information about what files and directories can be read by subjects.
- FJ: File directory for Java. The directories and files are related to Java environment.
- FD: File Delete using file.delete(). The permission to use the Java class method file.delete().
- NL: Network permission to connect Local client. The permission to connect the host which load applets.
- NR: Network permission to connect Remote host. The permission to connect the host that is not used by users.
- System information that should be Secret. The system information may cause danger and should be keep secret.
- SP: System information that can be Public. The system information without any potential threats can be kept public.
- EA: Executable program ACL. The access control list maintains the information about what system program can be executed.
- LOG: LOG file. The system log file records every important event in system.

### 3.2 The rules of our model

The diagram of our lattice model is shown in figure 1. The letter S denotes a security class of the subject, and the letter O denotes a security class of the object. The highest security classes are SA and LOG, so a system administrator can handle the whole system. The lowest security is BN2 that is a branch of the subject BN. It should be the safest state in the system, because it cannot touch any system resource.

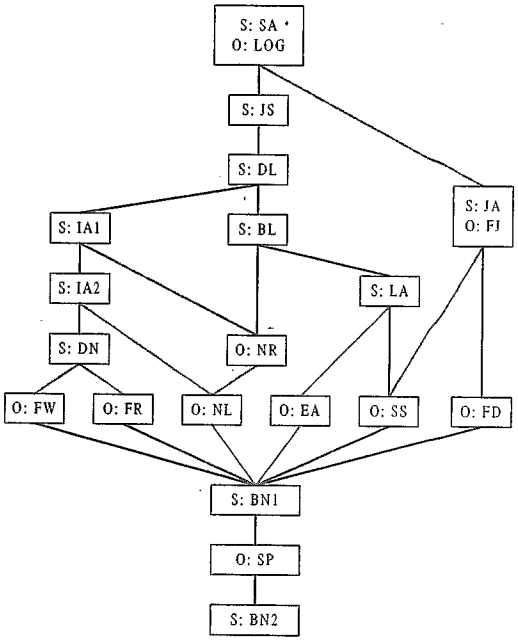


Figure 1: The proposed model

The procedures of information access in the traditional combined model are only reading from and writing to the storage, such as file system. While in a Java environment, we have to deal with more types of system resources including the network connection, the system information, and the executable system programs. Thus, we must expand the procedures to operate on all the four kinds of system resources. The rules of our model are as follows:

*(1) Subject s can write object LOG only if S(LOG)≥S(s).*

- Since LOG is in the highest security class, S(LOG)≥S(s) for each subject s∈SC. Thus, all activities must be recorded into LOG.

*(2) Subject s has the access right to object o only if S(s) ≥S(o).*

- The access right for FR is reading files. A subject that gains this object can read from the files and directories listed in the access control list.
- The access right for FW is writing files. A

subject that gains this object can write to the files and directories listed in the access control list.

- The access right for LOG is reading system log file. Since only subject SA has the same security level as LOG, SA is the unique subject that can read the record of system events in the log file.
- The access right for FJ is writing Java related files. In our model, only two kind administrator classes—SA and JA have this right to maintain the Java environment.
- The access right for FD is deleting files by using file.delete(). Since the process of deleting is a very dangerous action, only trustworthy Java standalone applications (JS), SA and JA can perform this operation.
- The access right for NL and NR is connecting host. NL is to connect back to itself, and NR is to connect with outside world through network. So the security class of NR is higher than that of NL.
- The access right for SS and SP is getting system information. The system information in SP is harmless, so all subjects except BN2 can access it.
- The access right for EA is executing programs. The subject that gains this object can execute the system programs listed in the access control list.

*(3) Subject s can modify object o only if S(s)=S(o).*

- Since SA and LOG are the highest security class, only SA can modify LOG. Similarly, JA is the unique subject that can modify FJ (files for Java).

### 3.3 Examples

In this section, we give two examples to illustrate how the proposed model works in a real Java environment. Example 1 exhibits the situation where users can lower the complexity of administration by using our model, and one can feel the user-friendliness in our design. Example 2 demonstrates that Java environment can be more secure by adopting our model since the information flow control can protect the system from unauthorized accesses.

**Example 1:** Alex wants to enjoy the Java world and executes a Java-embedded WWW browser. Two labels will be recorded. One label is assigned to BL for the applets loaded from the local file system, and another is assigned to BN1 for the applets loaded from the network. Suddenly, he finds an interesting game applet that provides multiple players with adventures in a puzzle, such as MUD. A dialog box pops up to allow the user to decide whether or not to allow that specific access, because an attempt is made by the applet to connect to an outside host and to write temporary files into the file system. Now, Alex can consider whether to upgrade the security class of the applet from BN1 to LA1 with the referential information that is displayed within the dialog box. If Alex trusts the applet or wants to take s risk to play it, he can choose yes to continue playing. Otherwise, he can choose no to begin another Java journey. Without our model, Alex must alter the security options one by one, and does not know what risks he may take. Even worse, he may loosen too much options to fall into a trap without even knowing it.

**Example 2:** Consider the following scenario:

- Alice executes an applet $A_1$ to simulate a virtual machine, and $A_1$ requests Alice to retrieve some secret system information for it. After $A_1$ can access the secret, it not only works more perfectly, but also saves the secret into a specified file.
- Later, Alice executes another applet $A_2$ to play a network bridge. Of course, $A_2$ must request the permission of network connection to communicate with other players. After $A_2$ can connect to the remote hosts, it not only communicates with the bridge players, but also searches the specified file and send the secret back to the applets programmers who writes $A_1$ and $A_2$.

This example demonstrates that the access control list is vulnerable to Trojan Horse attack. The Trojan Horse programs can get sensitive data by means but store or send to unauthorized users. With our approach, although $A_1$ can be permitted from BN1 to LA to access the secret information and store it into the file system. $A_2$ which is just upgraded from BN1 to IA1, cannot access the specified file whose class is the same as LA.

As discussed in the above two examples, we can point out what the advantages of our model are. Our model not only prevents the sensitive information from being send to the unauthorized subjects, but reduces the complexity of the administration.

## 4 Security analysis

Since Java is intended for distributed environments, we can consider its security from this aspect. The discussions are divided into three parts: access control, communication security, and authentication. The major difference between our approach and the current one is that the current access control scheme is totally based on the access control list method which is not secure enough and is rather complex to maintain, while we establish our access control scheme based on information flow control, and reduce the complexity of modeling a system by

reducing the number of classes with the access control list. To explain the superiority of our model, we first discuss the difference between access control list and information flow control, then we explain why our approach provides a more secure and easy-to-administrating Java environment.

### 4.1 The security analysis of distributed systems

Distributed systems must support a secure environment for users to compute and communicate. The importance of security in distributed systems grows as more and more sensitive information is stored and processed in computers, and transferred over networks. This implies the need for access control, communication security, and authentication.

Protecting system resources from unauthorized accesses is an important issue in distributed systems. In the current Java environments, the security mechanisms are based on the access control list, a variant of the access control matrix. The major disadvantage of the access control matrix is its vulnerability to the Trojan Horse attacks. Information flow control that our model based on can be demonstrated that no confinement problem exists, and is not susceptible to Trojan Horse attacks. Thus, our model provides a more secure environment. We will compare them in detail in sections 4.2 and 4.3.

In general, any sensitive information across networks should be keep secret. In a Java environment, there is no sensitive information in the Java code transported over networks. Even though two applets want to share a secure channel, it is the duty of applet programmers. If applets should not be able to access some sensitive information in a system, they cannot exchange it across networks with secure or non-secure channels. Thus, it cannot affect the system security.

Authentication is to make sure that a message is from the correct sender, and is received without being modified. In the design of a Java environment, we can use any well-known signature schemes to achieve authentication. For example, administrators can update the Java related classes signed by MD5 to prevent forgery or modification. This part of security is based on the security of signature schemes.

As mentioned above, our model replaces access control list with information flow control for access control. That is the key point why our model is more secure. To explain the advantages of our model, we first compare the two access control approaches, access matrix model and information flow control.

### 4.2 Comparisons between access matrix model and information flow control

The access matrix model provides a basic framework for protection based on the abstraction of operating system structures. The simplest and most natural representation is access matrix whose rows represent subjects, columns represent objects, and each entry consists of a set of access rights. A subject s can operate on an object o if and only if the access rights that appear in the entry contains the operation to be performed.

The direct implementation of access matrix model would be very inefficient and expensive since it is usually sparse. There are two popular implementation methods: access control lists and capabilities. The access control list approach which current Java environment bases on decomposes the access matrix by columns into several lists. Each objects has a list which records what operations by subjects are permitted. Similarly, capabilities decompose the access matrix by rows. Since the major improvement of them both focuses to solving the sparse problem, the access control list inherits almost the advantages and disadvantages from the access matrix model.

| | Access matrix model | Information flow control |
|---|---|---|
| Access control type | Discretionary | Non-discretionary |
| Confinement problem | exist | non-exist |
| Security | less secure | more secure |
| Major advantage | Flexible to add or delete objects, subjects, and operations | Monitoring the flow of information strictly and clearly |
| Major disadvantages | 1. Complex to analyze security<br>2. Vulnerable to Trojan Horse attack | 1. Hard to add operations and the class of objects, and subjects<br>2. Becoming complex with too many states and flow paths |

Table 1. The Comparisons between access matrix model and information flow control

| | Access control list | Information flow control | Our approach |
|---|---|---|---|
| Confinement problem | exist | non-exist | non-exist |
| Vulnerable to Trojan Horse attack | yes | no | no |
| Hard to monitor the flow of information strictly and clearly | yes | no | no |
| The same default permission for different class | yes | no | no |
| Complex to analyze security | yes | no | depending on the operating system |
| Security | less secure | more secure | most secure |
| Complex with too many states and flow paths | no | yes | no (reducing the number by access control list) |

Table 2. The Comparisons between our approach and the current Java approach

With the access matrix model, access control list, or capabilities, it is very clear that it is secure if trustworthy subjects are involved. However, untrustworthy subjects might cause some problems that cannot be solved by using them. For example, The threat, Trojan Horse, may cause access matrix model the confidential problems by copying secret objects, integrity problems by modifying files, and availability problems by deleting data. Since the access matrix model is vulnerable to attacks from Trojan Horse programs, the confinement problem that authorized subject can release sensitive objects to other unauthorized subjects exists in the access matrix model. Thus, access matrix model is not the best solution for access control.

Information flow control is an extension of the access matrix model to impose classifications on both subjects and objects. Because of the classification, unauthorized subjects would not be able to corrupt that data. Thus, there is no confinement problem by using information flow control. Table 1 summarizes the comparisons between access matrix model and information flow control.

Discretionary systems allow a subject to access an object at the discretion of the owner of the object. This means that each user can specify his own security options. Since objects, subjects, and operations can be easily inserted into or deleted from the matrix, it is easy and flexible to model a real system. However, it is complex to analyze the system security by scattering matrix, and is vulnerable to Trojan Horse attacks.

Non-discretionary systems allow a subject to access an object with the classifications of the object and the subject. This implies mandatory security rules are imposed on all users. Even if a Trojan Horse program can access privacy data, the data cannot be transferred out of the boundary enforced by the rules. Since the information flow control can monitor the flow of information strictly and clearly by the rules, it becomes a more secure approach. However, if there are too many states and flow paths, the system security will be complex and hard to analyze, too.

### 4.3 Comparisons between our approach and the current Java approach

Our approach is based on information flow control as the main architecture. To avoid the major disadvantage of the information flow control mentioned in the previous section, our approach incorporates the access control list to reduce the number of states or classes. Table 2 compared between our approach and current approach:

We discuss them into two parts as follows:

*(1) Security*

Since our approach is based on the information flow control, a subject cannot leak any secure information to other unauthorized subjects. Hence, our approach also solves the confinement problem in the approach of access control list.

Due to the resolution of the confinement problem, our approach is more secure than the approach of access control list. In addition to the scheme of the information flow control, we restrict the access right with the access control list. For example, to read a file, the subject needs to check not only such an information flow is permitted, but also the right to access the directory where the file is must be allowed in the access control list. Hence, our approach is more secure than that of information flow control, too.

To prevent the number of classes from growing rapidly, we adopt the access control lists into our model to

handle some system resources, such as file system and executable system programs. Thus, the drawback that is hard to analyze the security of the access control list would affect the security analysis for our model. For example, One authorized applet may save some secret in the files that are not protected by the operating system to unauthorized users who can read it in a non-Java environment. The problem can be solved by embedding the same security mechanism into the operating system. According to the announcement from Sun Microsystems Inc., almost all the operating system vendors have licensed and will embed Java into their operating systems in the near future.

As discussed above, our approach provides a more secure Java environment than any other existing approaches.

*(2) Simplification*

One of our main goals is to loosen the workload of administration. Since wrong administration would cause serious security problem in the system. Simplification is to provide not only a more user-friendly and easier environment for users, but also a potential direction to enhance system security.

In the current approach, users must alter all the restricted security options one by one to release system resource for applets. In our approach, users could just alter the security level from one class to another to release system resource for applets. In addition, users could get some suggestions or warnings to decide whether to alter the class of the applet or not when the applet requires more access right. Hence, our model simplifies the operations of system security for users and reduces the chance to fall into a trap without detection.

## 5 Conclusion

In this paper, we propose a new security model based on the information flow control to make the Java distributed environment more secure, and to simplify the operations of system security to provide a user-friendly interface. For typical users, they would like to have a more secure manager to loosen up with the simplified administration. For Java developers, they could enjoy the system resources that they need to code without worrying to leak any information to unauthorized users. For system administrators, it makes their jobs easier that all users in the system can work in a more secure environment. Our proposed model is the first one to achieve all these features.

## References

[1] R. S. Sandhu, "Lattice-Based Access Control Models," IEEE Computers, Vol. 26, No. 11, 1993, pp. 9-19.

[2] D. E. Denning, "A Lattice Model of Secure Information Flow," Comm. ACM, Vol. 19, No. 5, May 1976, pp. 236-243.

[3] R. S. Sandhu, "The Typed Access Matrix Model," Proc. IEEE Symp. Security and Privacy, 1992, pp. 12-136.

[4] J. A. Gougen and J. Meseguer, "Security Policies and Security Models," Proc. IEEE Symp. Security and Privacy, 1982, pp. 11-20.

[5] P. Terry, S. Wiseman, "A New Security Policy Model," Proc. IEEE Symp. Security and Privacy, 1989, pp. 215-228.

[6] A. Goscinski, *Distributed Operating Systems The Logical Design*, Addison-Wesley Press, 1991.

[7] D. Dean, E. W. Felten, D. S. Wallach, "Java Security: From HotJava to Netscape and Beyond," Proc. IEEE Symp. Security and Privacy, May 1996.

[8] "The Java Language Environment: A White Papers," Sun Microsystems Inc. (http://java.sun.com/whitepaper/javawhitepaper_1.html)

[9] "Security Features of Java and HotJava," OSF (http://www.osf.org/mall/web/SW-java/ security.htm)

[10] "Frequently Asked Quesetions - Applet Security," Sun Microsystems Inc. (http://java.sun.com/java.sun.com/sfaq/)

[11] "Overview of Java and HotJava," OSF (http://www.osf.org/mall/web/SW-java/intro.htm)

[12] D. Dean and D. S. Wallach, "Security Flaws in the HotJava Web Browser," November 3, 1995. (ftp://ftp.cs.princeton.edu/reports/1995/501.ps.Z)

[13] "HotJava: The Security Story," Sun Microsystems Inc. (http://java.sun.com/1.0alpha3/doc/security/security.html)

[14] "The Java Language Specifications," Sun Microsystems Inc. (http://java.sun.com/JDK-beta/psfiles/javaspec.ps)

[15] F. Yellin, "Low Level Security in Java," WWW4 Conference, December, 1995. (http://www.w3.org/pub/Conferences/WWW4/Papers/197/40.html)

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

[17] "CERT Advisory: Weakness in Java Bytecode Verifier," CERT, March 29, 1996. (ftp://info.cert.org/pub/cert_advisories/CA-96.07.java_bytecode_verifier)

[18] "Safe Internet Programming Research", Computer Science of department, Princeton University. (http://www.cd.princeton.edu/sip/)