行政院國家科學委員會專題研究計劃成果報告

行動式程式碼系統中軟體授權與保護機制

A Software Authorization and Protection Mechanism for Mobile Code Systems

計劃編號: NSC88-2213-E-009-017

執行日期 87 年 08 月 01 日至 88 年 07 月 31 日

計劃主持人:謝謝平

共同主持人:

執行單位: 國立交通大學資訊工程研究所

中華民國 88 年 07 月 31 日

Abstract

With the rapid development of Internet, software usage and development are changing day by day. In the past, a software developer distributes his software by selling channel, computer merchant, etc. And most of users only use this software in centralized operating environment, just like PC or workstation. Furthermore, due to poor distribution of software in the real world, most of users can't update their software and get the newest version immediately.

Today, through the Internet, any users can download the newest software and run it dynamically. They will no longer be asked to purchase the whole software when they just need to use part of the features. For software developer, they can always provide the newest software for users, and accounting the software usage times. This is called mobile code technology. But following the wider using of Internet, there comes a serious problem in software authorization and protection.

In the past, many approaches have been proposed to prevent software piracy, such as key disks, parallel-port locks, and custom serial-number validations. These schemes with authentication process embedded in the software cannot effectively protect the security attacks by a smart cracker. Once the software is cracked, it will be then distributed widely on the network.

This project is purposed for finding a practical method in software authentication and protection. In the beginning, we will survey the operating environment and clarify the operating roles in the mobile code system. Finally, we will intend to present a method to ensure the software authorization and protection for mobile code systems. On the other hand, for making it practical, we will also try to balance of "security" and "convenience" in our method.

摘要

網際網路技術的快速發展，漸漸改變了以往軟體的使用與散佈(distribution)的行為模式。傳統中，軟體開發者(software developer)所發展的軟體經由真實世界(real world)中的銷售管道散佈，如商家店面、訂貨機制等；軟體使用者，則大多只在單一環境(如個人電腦，工作站)上使用與操作軟體。更甚者，由於整個軟體散播管道不夠通暢，以致於大多使用者無法即時更新改版軟體。

當網路技術不斷推陳出新的今日，網路使用者已經能輕易的透過網路，閱覽即時文件與下傳應用程式碼；而軟體開發者透過網路，不但能統計與分析使用者的需求，更可以做到快速更新軟體版本、最短時間內散播最新開發軟體等優點。此外，像 Java 這類新世代語言寫成的軟體，也開始改變軟體的使用方法，人們在任何地方將只需要自網路上下載所需的程式碼，即可執行他們想要的軟體；而軟體商也可依使用者的使用次數（甚至是選用的功能項目）來收費，或即時更動最新版本。像這類軟體即是所謂的行動式程式碼（mobile code）。但是，因為網路的便利與無所不在的特性，反而使行動式程式碼組成的軟體比傳統軟體面臨更大的破解、盜拷等安全威脅。

傳統的軟體授權與保護技術，如序號(serial number)保護、母片保護、硬體鎖等，在此種新的網路散播管道下，都受到嚴重的挑戰。今日一旦軟體放上網路，往往被使用者大量的複製、無授權的使用。即使這些軟體有使用到傳統的保護措施，聰明的軟體破解者也能的利用網路將軟體破解的方法快速的傳播出去，造成軟體授權制度上的更大傷害，以致於軟體開發者對於普及的網路反而裹足不前。

本計畫即是針對未來的行動式程式碼系統的使用環境，找出一個可行的軟體授權與保護的方法。在此計畫中，我們將先對行動式程式碼系統的運作環境作相關資料的收集與研究，以釐清整個環境相關角色的運作模式（這是因為目前都還沒有公認實用的行動程式碼的運作環境，所以我們必須先予以合理的假設，以利日後的研究。）接下來則是針對此類軟體，提出可以保證軟體安全的使用授權方法與保護機制。同時在此研究案中，我們會特別考慮到「安全性」與「使用效率」的雙重因素，因為一個只是基於安全保護的軟體，可能導致執行效率低落、使用不便的缺點。相對的，此安全機制被軟體製造商採用的機會也就大打折扣了。

Table of Contents

List of Figures

Chapter 1

Introduction

With the fast development of Internet, network transmission speed grows faster and faster. Advancements in network technology allow the network users to do a lot of jobs that were difficult to be accomplished in the past. Due to the evolution of network technology and the interest of market, large scale distributed systems are becoming of paramount importance. The growing importance of telecommunication networks has stimulated research on a new generation of programming languages. Recently, mobile code languages [Ghezzi97][Gosling96][Gray95][Hall96][Jaeger96] have been proposed as a technological answer to the problem. Such languages view the network and its resources as a global environment in which computations take place [Bic96][Carzaniga97][Ciancarini97][Nog96][Perret96].

1.1 Background

The development of World Wide Web combines many traditional services and lets every user to navigate the whole Internet using a single Web browser. In mid-1995, Sun Microsystems announced the Java [Gosling96] language. The Java language is a simple, object-oriented, portable, and robust language that supports mobile codes. Java augments the present WWW capabilities by dynamically downloading the mobile code fragments, called applets, and running this code

fragments locally.

The development of mobile code technologies changes the style of software usage. The mobility and cross-platform characteristics of Java language allow software rental on the network. For users, when they want to execute some functions of the software, they can download the newest software across the network and run it dynamically. They will no longer be asked to purchase the whole software when they just need to use part of the feature. Revision for software in the environment becomes simple. For software developers, they can always provide the newest software for users, and they know how many times the user has downloaded software. Software rental requires a good software authorization and protection model to prevent unauthorized use.

There are several projects for supporting authorization-based access control in the WWW environment [Samarati96][Kahan95], that protects the information in hypertext systems. Java applets can be integrated as a part of the hypertext system, however, the current authorization models for WWW are inadequate for protection of Java applets. There are many differences between Java applets and common documents in a hypertext system. Documents in a hypertext system contain only data, which displays on the screen at the host of the browser. Java applets contains both code and data, and the state of browser may be different for different users in different locations. Users without restriction can execute some code, but some code has to be executed under control. There are many situations that the code has to be executed under control. For example, sometimes we want to know how many times

a user executes the code and which function of the code the user executes.

1.2 Software Security and Protection

Although the rapid development of network and advanced technologies enable new software capabilities and wide market interest, the software piracy is still a serious and tough problem for a long time. Various software protection schemes have been proposed, but a malicious user can easily crack some schemes, and some require additional costs for users.

Software security

The problem of software piracy causes considerable losses to software vendors [Curtis94][Neff94]. Copyright laws [Donovan94][Darkin95] regarding software are rarely enforced, thereby causing major losses to the software vendors.

A serious problem in the authorization model of a hypertext system is that the data can be copied or disseminated [Samarati96]. A user accessing a document can copy the document, and then disseminate it without the permission of its owner. This problem happens because the authorization is only done on server, once the user receives a document, he becomes the owner of this document and has unlimited privileges on it.

Such problem appears to be even more serious in software applications.

Software piracy is the unauthorized copying, use, or distribution of software products, and this problem becomes more serious on the network environment. Here are several basic software piracy problems [Curtis94]:

Illicit dissemination: Making extra copies of the program and disseminating to other unauthorized users.

Unauthorized use: Unauthorized use of software applications designed for restricted use such as executing for a period of time or a number of times.

Counterfeiting: The illegal duplication and sale of copyrighted software, often in a form designed to make the product appear legitimate.

To deal with the problems of the software on the network, not only the software itself but also the environment associated with the software must be considered. The modified version of software is be harmful to users executing it, since it may contain a Trojan horse or a Virus [Barker89][Dean96][Rubin95]. The malicious code that contains a Trojan horse or a virus accessing user's system resources such as the file system, the CPU, the network, and the graphics display may cause unpredictable effects from stealing user's privacy to damaging resources in user's environment. Besides Trojan horse and virus, a user who modifies the code to deviate from the prescribed execution may cause more problems to other parties on the network. For example, a user may cheat in a multi-player game on the network if he has the ability to modify the prescribed code of the software. Some execution results of the

software might be supposed to write to a database server on the network, and a user can modify the code to stop the writing process or forging fake results. Therefore, such problem must also be considered in the software protection on the network.

Software protection

There have been many approaches to stopping software piracy [Wilson97], ranging from key disks [Voelker86] to parallel-port locks. Most of these software protection schemes embed access control mechanisms in the code, and a user has to pass these authentication processes before using the software. The process may require the serial number of the corresponding user, password from the manual, or checking the source place where the software is located (CD or floppy disk, for example). Many crackers have cracked these authentication processes. The difficulty to crack such protection scheme depends on how complexity this part of code is written. For example, some software vendors put checksum values for the authentication process in the software, if someone tries to modify the code to bypass the authentication process, an error may be found in the future and the execution will be terminated. This adds the difficulty to crack the software. However, it is not a long-term solution that prevents from unauthorized use but just increases the time to protect from been cracked.

- Authentication process may check key disks, parallel-port locks, or custom serial-number validations

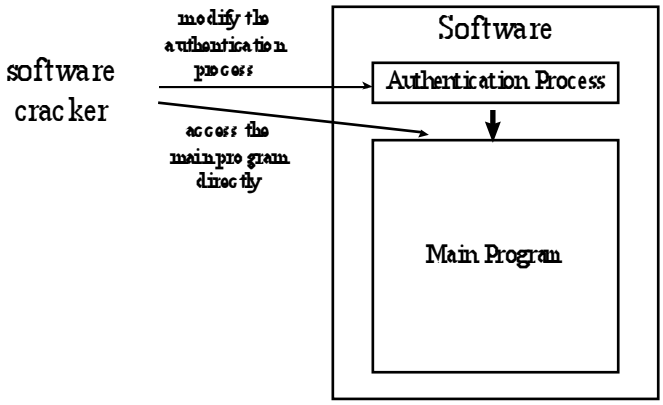


Fig 1.1 Common software protection schemes

Problem of illicit dissemination of software appears to be more serious on the network. We wish to control the access that only authorized users can download and use the software. Before a user downloads the software, we can easily control the access to allow only legal users to have the download permission. However, after a user gets the applet, it's difficult to control the permission. Therefore, common software protection schemes that combine the authentication processes in the software itself cannot effectively protect the software from been cracked by a smart cracker.

A practical software protection should ensure that illicitly duplicating or unauthorized use of the software is at least as hard as rewriting it from scratch [White90]. A purely software-based solution is almost impossible, since any software is just a binary sequence. Stronger schemes have been proposed to protect

the software with the help of tamper-resistant hardware devices, which consists of physically shielded CPU and memory [Best79][Kent80]. However additional costs of these schemes often discourage users to purchase the software.

1.3 Contributions

In this project, a software authorization and protection model for mobile code systems is proposed. To achieve flexible and global security for the rapid growing network environment, the protection for both the software property and principles in the network environment have been taken into consideration. In the model, the privileges to access the applets in software are separated and distributed to a number of trusted principles called trusted computational proxies. Dependent applets are distributed to different proxies, which leaks little information in the case that a proxy is compromised. In this environment, an optimal assignment of applets is also proposed to minimize, under the security constraint, the computation load of the proxies and the communication load between proxies and users.

1.4 Synopsis

This report is organized as follows, chapter 2 gives the related work regarding software protection methods, Java language, and RMI technologies. In chapter 3, our proposed model for software authorization and protection is presented, which is

based on the concept of separation of execution privileges. A model for software partitioning to achieve protection in this environment is presented in chapter 4, and related issues for performance and security are concerned. Finally, we give the conclusions in chapter 5.

Chapter 2

Related Work

In this chapter, we present some related software protection schemes, Java language, and RMI (Remote Method Invocation).

2.1 Software Protection Schemes

There have been many approaches to stopping software piracy, ranging from key disks to parallel-port locks. Most software protected with these schemes has been broken easily. Once they have been broken, the software was then spread widely. Since the above schemes protect the software from only naive users instead of smart crackers, stronger protection schemes must be designed. In this section, we present two stronger software protection schemes, which protect the software based on tamper-resistant hardware.

2.1.1 ABYSS: Architecture for Software Protection

ABYSS (A Basic Yorktown Security System)[White90] is an architecture for protecting the execution of application software. It can be used as a uniform security service across the range of computing systems. In ABYSS, applications are partitioned into protected and unprotected processes. The protected application

processes are executed in a secure computing environment called a protected processor. Execution of the applications are determined by a logical object called Right-To-Execute. The protected processor enforces the access control with respect to the Right-To-Execute for applications. Authorization process of Right-To-Execute can be carried out by tokens, which are introduced as a new use-once authorization mechanism and useful when authorization are distributed physically.

To protect an application, the software vendor must create a part of the application to be executed securely, encrypt it, create a corresponding Right-To-Execute, and create an authorization process for installing that Right-To-Execute. The same ABYSS processors, which execute protected applications, can be used to perform the critical steps in this process, so no special development systems are needed.

2.1.2 Software Protection and Simulation on Oblivious RAMs

A machine is oblivious if the sequence in which it accesses memory locations is equivalent for any two inputs with the same running time. In this paper [Goldreich96], the key problem of learning about program from its execution has been formulated, and the problem of software protection is reduced to the problem of on-line simulation of an arbitrary program on an oblivious RAM. It provides theoretical treatment of software protection.

It is showed that if the one-way functions exist, this software protection scheme is robust against a polynomial-time adversary who is allow to alter memory contents during execution in a dynamic fashion.

2.2 Java Language and Remote Method Invocation

Mobile code technology enables the code to be downloading dynamically from a remote server and executed in the local machine. Here we present the Java language, which is most popular mobile code technology in the current Internet environment. The RMI (Remote Method Invocation) which enables cooperation between machines on the network in the Java environment is also discussed.

2.2.1 Java Language

Java applications, or applets, are different from ordinary applications in that they reside on the network in centralized servers. The network delivers the applet to your system when you request them.

The Java language changes the passive nature of the Internet and WWW by allowing architecturally neutral code to be dynamically loaded and run on a heterogeneous network of machines such as the Internet. Java provides this functionality by incorporating the following features into its architecture. These features make Java the most promising contender for being the major protocol for the

Internet in the near future.

2.2.2 Remote Method Invocation

Distributed systems require that computations running in different address spaces, potentially on different hosts, be able to communicate. The development of Remote Method Invocation (RMI)[Sun96a] enables software developers to create distributed Java-to-Java applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. With the RMI technology, a Java program can make a call on a remote object once it obtains a reference to the remote object, either by looking up the remote object in the bootstrap-naming service provided by RMI, or by receiving the reference as an argument or a return value.

The Java remote method invocation system described in this specification has been specifically designed to operate in the Java environment. While other RMI systems can be adapted to handle Java objects, these systems fall short of seamless integration with the Java system due to their interoperability requirement with other languages. For example, CORBA presumes a heterogeneous, Multilanguage environment and thus must have a language- neutral object model. In contrast, the Java language's RMI system assumes the homogeneous environment of the Java Virtual Machine, and the system can therefore take advantage of the Java object model whenever possible.

Chapter 3

The Proposed Authorization and Protection Model

As the speed of network transmission becomes faster and faster, many jobs tend to be processed by tightly connected computers. A network-computing environment can be established by the mobile code technologies. In this environment, the computational and storage resources may be spread on different locations instead of a single computer.

In mobile code systems, the software is composed of many applets. The applet is a piece of software that can be downloaded dynamically from the remote machine and executed in the local machine, and the cooperating of these applets can process the job. In the environment, an authorization and protection is proposed which provides security for the software by delegating some critical execution services to a trusted and protected party.

3.1 System Model

The execution of software can be viewed as the following three parts, incoming flow information, transformation process, and outgoing flow information. For an outgoing information, if the information flows through an applet, we can say that this

applet participates in the transformation process for the outgoing information. Therefore, if we remove some applets that participate in the transformation process, the execution will be stopped without help of the missing applets.

3.1.1 The Proposed Protection Model

With the RMI (Remote Method Invocation) technology for Java language that enables cooperating of computers on the network, we proposed a model that protects the software with the help of a trusted, protected computational proxy servers instead of tamper-resistant hardware devices installed in the user's environment. In this model, applets of the software is partitioned into general and privileged applets. The users can get only general applets and the privileged applets will be forced to be executed in a protected environment.

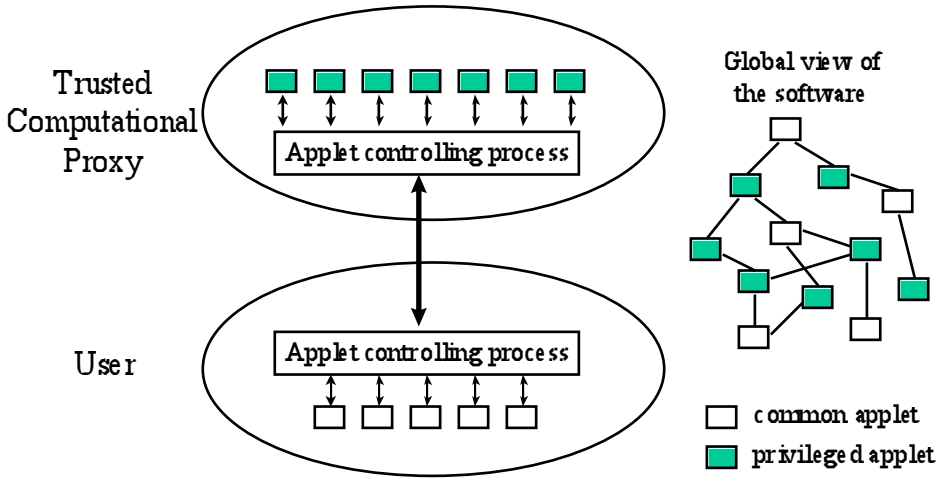


Figure 3.1 The proposed protection model

The trusted computational proxy provides computation services for privileged applets. Only a trusted proxy has the capability to get privileged applets and execute them. The proxy executes the applets and returns the result to the user. Since some of the applets are forced to be executed in the proxy server, an unauthorized user cannot benefit from the software with only part of the applets. There can be more than one proxy, and each proxy executes part of privileged applets. Then the compromise of one proxy will not leak all privileged applets. In the proposed model, applets to be downloaded are encrypted by applet keys, and the applet keys for each applet are different. These keys are only available to trusted proxies or authorized users.

System Components

In this model, there are six major components:

Software Vendor: The company who develops the software.

Certificate Authority: The party who issues public and private keys.

Software Authentication Center: An accredited organization that authenticates the software developed by software vendors, and signing legitimate parts of the software.

Applet Server: The server who stores applets provided by software vendors. When a host wants to execute an applet, it first downloads the applet from an applet server.

Trusted Computational Proxy: The server that provides computational services of privileged applets for users.

User: The user who uses the software.

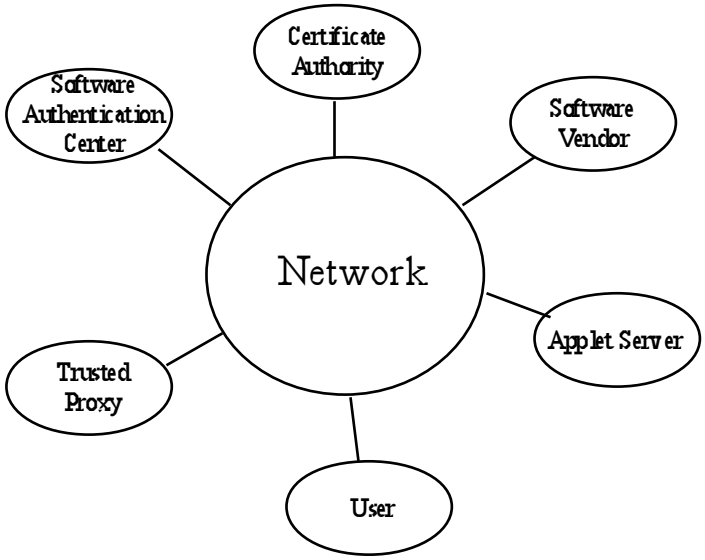


Figure 3.2 System components

3.1.2 Licenses for the Software

In our model, there are two kinds of licenses, publication license and execution license. The publication license gives the right for software vendor to distribute an applet and the execution license gives the right for user or proxy to download and execute an applet.

Publication license

For each applet, there is a publication license associated with it. The publication license is issued and signed by software authentication center, and every applet provided by software vendor, which must have a legal publication license.

A publication license consists:

1. Serial number
2. Software vendor information
3. Software authentication center information
4. Applet information (ID, version)
5. Message digest of the applet (optional)
6. Issuing and expiration time
7. Other information

The license is signed by the center's private key. When a user or a proxy downloads an applet from the applet server, it verifies the applet by the center's public key and also checks the message digest and expiration time of the applet.

The message digest of an applet is optional. For some applets, we give the message digest to an authorized user in another way instead of placing it in the publication license. This helps to reduce unauthorized use for the applets. We will discuss this later.

When the software vendor releases a new applet, it first sends it and the related information about the applet (for example, specification or source code) to the

software authentication center. The software authentication center checks the applet, and if there is no problem with it, the center issues a publication license of this applet and sends back to software vendor.

For some applets, sometimes it is not easy to verify them in the first time. In this case, the center can set a shorter expiration time in the publication license for such an applet. It issues new licenses periodically to the software vendor and will stop the process if any problem of the corresponding applet found in the future. Then the software vendor updates new publication licenses on all applet servers see Figure 3.3. The expiration time for each publication license depends on the policy for the software authentication center. Generally, longer expiration time can be assigned if an applet from a software vendor is more trusted or easier to be verified.

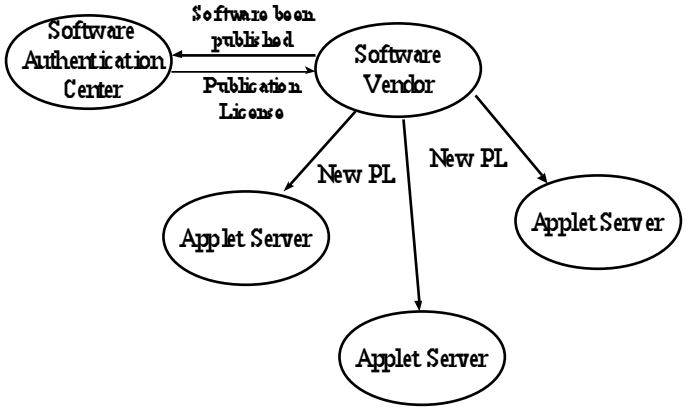


Figure 3.3 Publication license updating

Execution license

The user or proxy must get an execution license to execute the corresponding applet. The execution license is issued and signed by software vendor.

The execution license consists:

1. Serial number
2. Execution capabilities for applets of the software
3. Delegation capabilities for applets of the software (For user only)
4. User or proxy's information
5. Software vendor information
6. Issuing and expiration time
7. Other information

The execution capability of an applet determines whether a user or a proxy can download the applet. The delegation capability determines whether a user cans delegation the execution of an applet to the proxy. Delegation here means delegate the execution to a trusted party if a user cannot execute it directly. If the user has the execution capability of an applet, he can get some extra information of the applet from software vendor, for example, applet key or message digest of the applet. To execute the privileged applets that have to be executed in the proxies, a user must have the delegation capability for these applets.

The execution and delegation capabilities for a user dependent on how many applets the user has been authorized to use. If the user is interested in only some features of the software, software vendors can issue the license with the capabilities for only the applets providing these features.

3.1.3 Using the Software

The user can purchase the execution license to the software he interested in from the software vendor. Once the user received the execution licensed offered by software vendor, he can begin to use the software. In this section, we describe the related issues when a user is using the software.

Applet Downloading

In the mobile code system, the applets are dynamically downloaded from a remote server and executed in local machine. Applet downloading is necessary before execution if there are no previously cached applets in a proxy or user's computer. The applet server controls the access for those applets to be downloaded. The client (proxy or user) sends the request of the applets needed to be execution associated with his execution license. If the license consists of the execution capability with respect to the applet and the license is valid, the request will be accepted, and otherwise it will be denied.

When a user or a proxy received an applet from the applet server, he can decrypt it with the corresponding applet key. The verification process verifies the validity of an applet, which includes the correctness and the effectiveness of the downloaded applet.

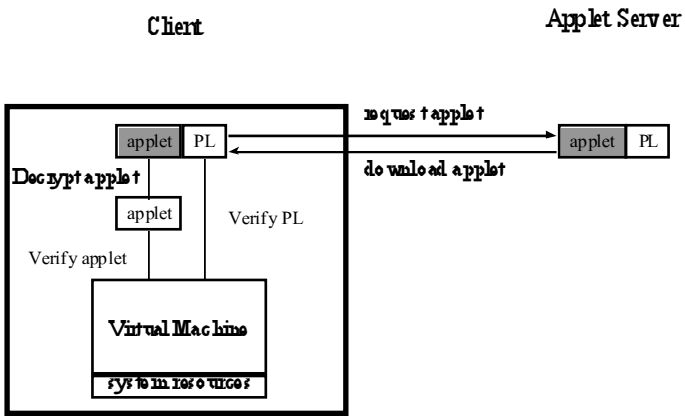


Figure 3.4 Applet downloading and verification

Execution of the software

After a user downloads the applets from an applet server, he can begin to execute them. Since privileged applets are forced to be executed by the proxies, therefore the user has to bind these applets first before execution. In the binding phase, the user sends his execution license to the proxy server he wants the execution to be delegated. Both user and proxy mutually authenticate the execution licenses with

each other. The execution then proceeds by executing the applets corresponding to the capabilities listed in user's execution license.

If there are more than one proxy participated in the execution, the user will be required to explicitly make connections to each of them and authenticate with each other.

At the first time for using the software, the user asks to software vendor for a list of available proxies. Then he chooses the proxy for computational service and register himself at this proxy. Registration for execution licenses will be discussed in the next section.

3.1.4 License Registration and Revocation

Once an execution license has been issued to a user, the user can use the software with the capabilities listed in the license. However, sometimes the software vendor may wish to revoke the license for a user if illegal behavior of the user has been found. Moreover, with the registration and revocation, the license can be easily expired by the number of executions. The proxy records the number of executions for the user, and if it exceeds the limitation described in the user's execution license, further execution will be refused.

Registration

Registration is required for the first time when a user wants to use the service provided by a proxy. The execution license will only be valid for the proxy if there is a corresponding registry $\overline{U\ P\ SN_U}_{D_S}$ (D_S is software vendor's private key) in the proxy. When a user wants to delegate the execution to a proxy, the proxy checks both the user's execution license and the registry. The license without a corresponding registry will be considered as invalid. The following is the steps for registration:

Steps 1: User sends a request associated with his execution license to software vendor for registration at a proxy.

Step 2: Software vendor checks the validity for user's license. Go to Step 3 if the user's license is valid otherwise stop.

Step 3: Software vendor sends a message $U\ P\ SN_U$ (signed by software vendor) to the new proxy for adding user's record at the proxy, where SN_U is the serial number for the license.

Step 4: Software vendor updates its own registry for the user.

Revocation

To revoke an execution license in a proxy, the software vendor simply tells the proxy to remove the registry for the user and then removes the registry located in the software vendor itself. Then the user's execution license will be revoked because no

registries can be found on the proxy.

3.2 Illicit Dissemination and Unauthorized Use of the Software

A serious problem in software protection is the illicit dissemination of the software. As we mentioned earlier, such problem is even more serious in mobile code systems. Although the applets in the applet server are encrypted, an authorized user who gets the applet may decrypt and illicitly disseminate it to another unauthorized user. Moreover, a trusted computational proxy may also be compromised. The more applets a user can get, the more information he will be able to gain from the software. However, it is very difficult to control the software once it has been illicitly distributed, since the software is just a binary sequence of data. Common protection schemes with authentication processes embedded in the software failed to stop the problem from smart crackers. Therefore something must be done to reduce the problem that the applets of authorized users or proxies illicitly disseminated to unauthorized users.

3.2.1 Discouraging Illicit Dissemination

Common methods preventing or reducing such problem use the technique of digital watermarking. With this technique, software vendor can hide some embedded data into the software for the purpose of identification and copyright, in

order to discourage unauthorized copying and dissemination. Many papers have been proposed for this area [Bender96] [Berghel96] [Brassil95] [Choudhury94]. It is useful for tracing the software that has been illicitly distributed. When the uniquely marked ownership for a consumer is embedded in the software, he tends to be unwilling to distribute the software to the network because the copyright violation may be found by software vendor.

3.2.2 Discouraging Unauthorized Use

Here we propose a scheme to reduce the problem of software piracy in another point of view by discouraging unauthorized users to use the illicitly disseminated software instead of discouraging illicit dissemination.

The modified version of applet is be harmful to users executing it, since it may contain malicious code such as a Trojan horse or a Virus. The malicious code accessing user's system resources such as the file system, the CPU, the network, and the graphics display may cause unpredictable effects from stealing user's privacy to damaging resources in user's environment. Therefore, users usually refuse to execute unknown or untrusted code from the network. In JDK 1.1 (Java Development Toolkit)[Sun96b][Gong97], the code signing feature is provide and the user who downloads the applet can verify the code signed by the author. If the applet is not trusted, execution will be restricted in a sandbox with only limited system resource provided, as shown in Figure 3.5.

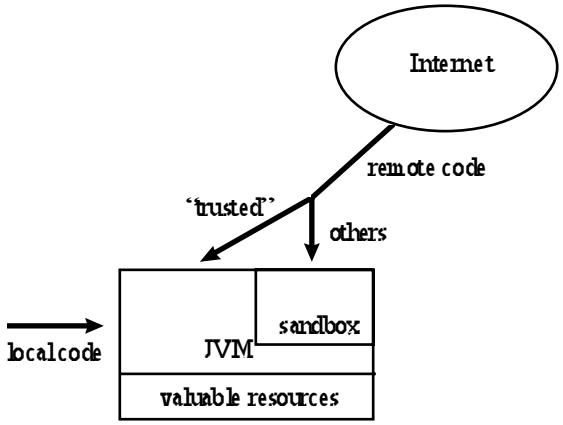


Figure 3.5 JDK 1.1 Security Model

Based on this nature, our proposed scheme discourages the unauthorized use in mobile code systems by preventing unauthorized users to verify the applets downloaded from the network. In Figure 3.5, the user tries to download the applets from the applet server without an execution license will be refused. However, an authorized user may disseminate the applets he has to the unauthorized user. We apply the technology of undeniable signature [Chaum90] to make an authorized user not been able to proof that his applet is valid to other users. The software vendor or software authentication center cannot deny those applets signed by them and has to be responsible for the applets if any problems due to the applets are found in the future.

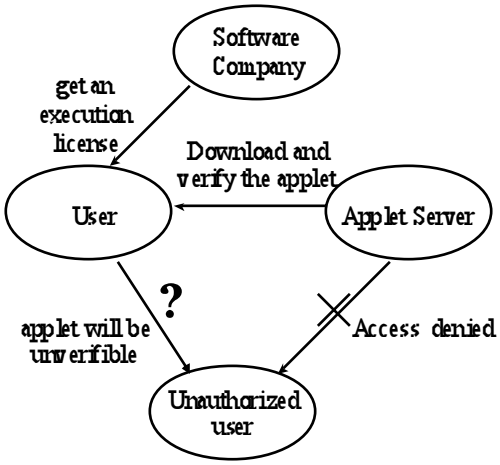


Figure 3.6 discouraging the unauthorized use of software

Nontransferable Message Digest

In our scheme, a user cannot convince to another user that the message digest he has is valid. The unauthorized use of software is discouraged by the nontransferable message digest, which applies the undeniable signature scheme proposed by [Chaum90]. The protocol for requesting a nontransferable message digest is shown in Figure 3.7. A large prime, p, and a primitive element, g, are made public, and used by a group of signers. The signer has a private key, x, and a public key, $g^x \bmod p$. For a message m, the signer first computes $z = m^x \bmod p$. A user has to get the signature with his execution license. If he has the execution capability for the applet, he can get the corresponding message digest from the signer.

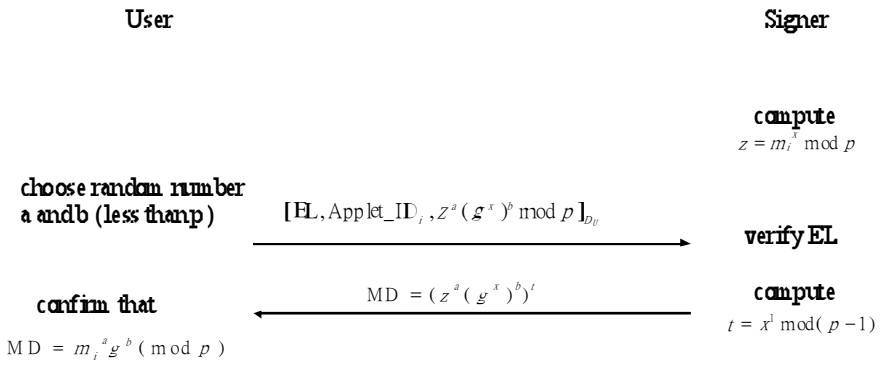


Figure 3.7 Nontransferable message digest

3.3 Security analysis

In this model, the software may be compromised if

1. The proxy server is compromised

If a proxy is compromised, all privilege applets in this proxy will be also compromised. So a proxy has to be trusted by the software vendor and has to be secure from been cracked. A proxy can be software vendor itself or another trusted party. There can be several proxies, and each proxy owns execution licenses for a part of privileged applets. Such that the software can be secure if not all proxies are compromised.

2. User can reconstruct the protected part of applets successfully.

Although some privileged applets are executed in the proxy and other applets executed by user have to dependent on the other privileged applets, a user still can try to crack the software by reconstructing the missing applets. In chapter 4, we proposed a model for software partitioning, which offers a way of partitioning applets into common and privileged parts by reducing the chance for a user to benefit from the software with partial applets.

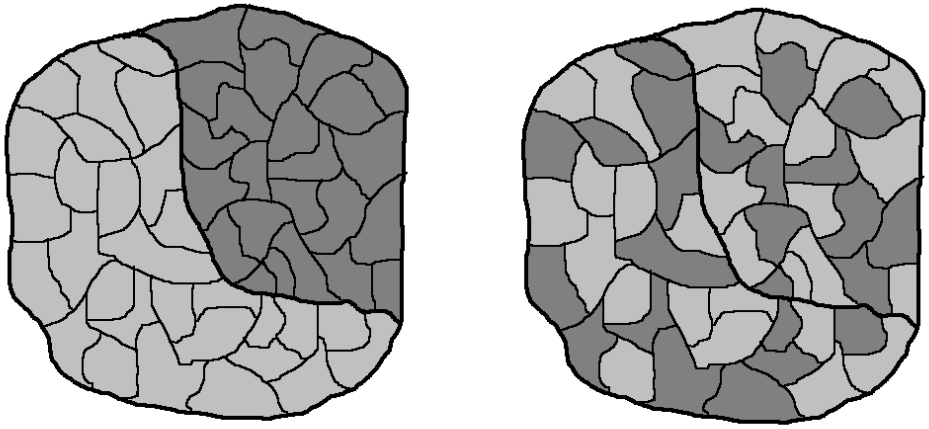
Chapter 4

Software Partitioning

Software partitioning means separating applets such that a user can not benefit from the software holding only part of applets. Our goal is to partition the software in such a way that if the user gets one applet, he will not be able to get an acceptable result from the applet if it requires the help of the execution of other applets.

In Figure 4.1, we compare two different ways of software partitioning. Assume that the software is represented in the graph which each separated part means a code fragment, and the execution of a code fragment depends on the execution of adjacent code fragments, that is, there will be procedure calls or invocations in the execution between the adjacent code fragments. If there are two main modules in the software, and the light color in the graph means the code fragments a user can get, and the dark color means the codes that have to be executed in a protected environment, that is, the trusted computational proxy. It is straightforward that the right part of partitioning provides better protection for the software than the left part. The method of partitioning in the left graph merely gives all code fragments in the first module to the user and all code fragments in another module to the proxy. An unauthorized user who gets these code fragments can access the first module of the software and get some partial results. In the right graph, an unauthorized user can still get as many code fragments as in the left graph. However, he will not be able to benefit from any one of the two main modules in the software, because many of the code fragments he

got dependent on the execution of other code fragments in the proxy.



Poor partitioning Better partitioning

Figure 4.1 Example of partitioning

The execution of one applet may give some information to the user. The more applets the user can get, the more information may be gained from the user. If the user gets all the applets, we can say that the whole software is compromised. However, for two nonadjacent applets, since they are not directly dependent, the user cannot gain more information directly from them unless he can also get the applets that have dependencies between the two applets and obtain a larger subset of dependent applets. For two applets, the user will get more information from them if they are directly dependent.

4.1 Proposed Model for Partitioning

Software in the mobile code system can be represented as an undirected graph $G = (V, E)$, where the vertices are the applets and the edges mean the dependency between any two applets. If an applet may invoke another applet in the execution session, we can say that they are dependent. For two dependent applets, there will be messages passing between them in the execution of the software.

In the software, we assume that user can get more acceptable result from it if he can get a larger subset of the connected applets. Giving user two independent applets will provide better protection than two dependent applets, because the user cannot benefit from two independent applets directly if they dependents from other applets executed in the proxy. Based on the assumption, we proposed a partitioning model, which all any two applets executed by the user are independent, as shown in Figure 4.2.

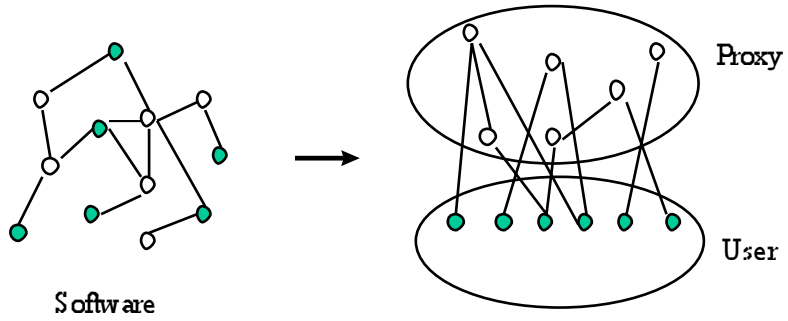


Figure 4.2 the proposed partitioning model

Each of them depends on the execution of other applets executed by the proxy to gain the result of a larger subset of applets. Our proposed model considers the security only to the applet level, that is, we do not look inside the applets and concern how it is written. An applet is a basic element in our model. For small software with only several applets, a heuristic partitioning may work better. For large software composed of many applets, our model gives a good protection by partitioning the software into minimal piece of independent code fragments.

Based on this model for assigning independent applets to achieve security, we consider the following two issues:

1) Performance

2) High Security

In the first consideration, we wish to achieve good a performance considering the computational load of the proxy. Since the proxy provides the computation services for users, the load may become heavy. Therefore partitioning in such a way that finds a better performance in this environment becomes very important. Under the constraint that all applets executed by the user are independent, we want to assign as more applets in client as possible. If each applet has a different computational cost, we can find an assignment that minimizes the computation load for the proxy. Later,

we will also take the communication load between any two applets into consideration, and find the optimal assignment for both the considerations for computation cost and communication load.

In the second consideration, we want to assign applets in such a way to achieve higher security. If we want to minimize the information the user can get, we can assign as more applets in the proxy as possible. Since the user gets only the result of the execution from proxies, higher security can be achieved. In this case, we also consider that the proxies may be compromised.

4.1.1 Assignment for Applets

In the software represented by a graph $G = (V, E)$, the applets executed by the user is assigned as number 0, and number 1 or greater represents the applets executed by the proxy.

Not all applets of the software will be freely assigned. Some applets may have special properties and have to be assigned in specific locations. Before partitioning, we find this kind of applets and assigned them first. The steps for initial assignment are described as follows, and an example is shown in Figure 4.3.

Step 1: Mark the nodes that have to be placed in specific locations.

There may be some applets that have to be executed in specific locations. For example, some applets may be designed for reading data from user's

keyboard, displaying something to the user's monitor or reading/writing something from user's hard disk. This kind of applets has to be executed by user, and assigned as number 0. Some applets have to open some network connections from a proxy (in a firewall, for example) or reading/writing something from proxy's file system. These applets should be placed in the proxy, and assigned as number 1. In this step, all special nodes are marked, as a number depending on the location the applets has to be assigned.

Step 2: Assign 1 to the nodes adjacent to nodes assigned as 0.

Since the applets executed by the user must be independent, all applets adjacent to applets assigned, as number 0 cannot be assigned as number 0 again. These applets have to be assigned as number 1.

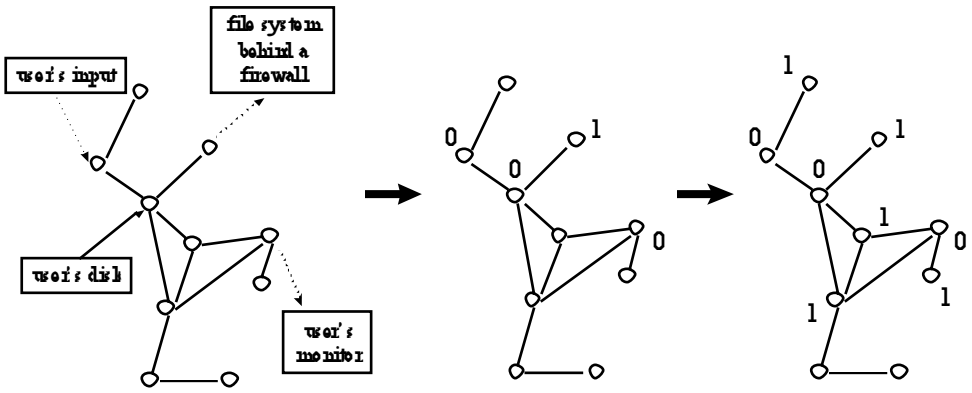


Figure 4.3 An example for initial applet assignment

Here are some examples for this kind of applets that have to be initially assigned.

User: Reading from keyboard

Reading or writing from user's hard disk

Display on the monitor

Communicate with network with user's identity

Proxy: Reading or writing from proxies files systems

Execution from behind the firewall

Applets consists of critical codes

In addition, the security concerns is also an important factor for the initial assignment of applets. In the network environment, sometimes an applet may do some operations to a specific principle, and the correct execution has to be assured. For example, the software vendor may want to record the state of the execution of software provided to users. Sometimes a trusted party can only access a database. If the user performs the execution of the applet, he may modify the code to deviate from prescribed execution and creates faulty results. Thus such applets have to be assigned to the proxy to ensure correct results. The delegated computing ensures that the critical part of code is correct been executed, as shown in Figure 4.4.

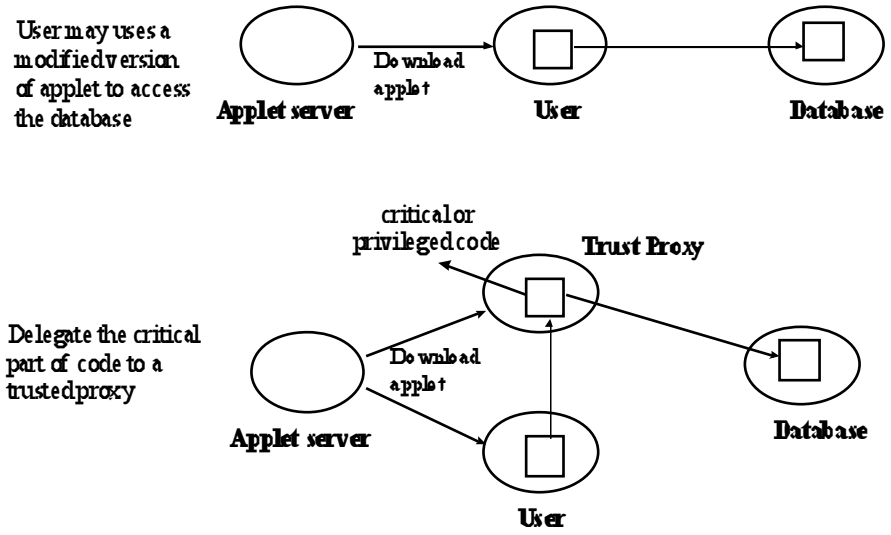


Figure 4.4 Delegated computing

4.1.2 Complexity of an applet

The execution of software can be viewed as three parts, incoming flow information, transformation process, and outgoing flow information. In the transformation process, some applets cause little influences to the outgoing information and some may just pass the information flowing to them without any transformation. For these applets, assigning them to the proxy will be little use since an elite user may reconstruct the missing part of applets easily. If an applet is not written with enough complexity, it can be assigned to user. An applet executed by the proxy should have enough complexities described as follows [White90].

Semantic Complexity:

Semantic complexity reflects the difficulty of reconstructing the protected part by examining the environment of its interaction with the unprotected part. At one end of a spectrum of partitioning methods, selected obscure parts of the application are protected. For instance, an application may contain a proprietary algorithm, all of which could be protected. If that part of the program was difficult to write initially, it may be difficult for an attacker to reconstruct it. At the other end of this spectrum, random parts of the application could execute in the protected environment. This is semantically complex to the extent that it is difficult to understand a program that has a large number of lines missing.

Combinatorial Complexity:

Combinatorial complexity reflects the difficult of exhaustively characterizing the behavior of the protected part by watching what it does. Consider an application in which there are n access points in the unprotected part, at which accesses are made to the protected part. At each access point, a k bit argument is passed to the protected part, and the protected part performs some calculation. If this results in $\Omega(E^K)$ independent states of the system, essentially all possible values of the argument must be tried by an attacker to completely characterize the effects of the calculation. This is the case, for instance, in a one-to-one function of the argument, whose value is

returned by the calculation. The characterization can be made even more difficult if some or all of the results are stored in the protected part instead.

4.2 Partitioning For Performance Considerations

If the proxies are trusted and protected, adjacent applets in the proxy will not be a problem. We just need to assign the nodes that all applets in the user are independent, because the unauthorized users cannot access applets in the proxies.

It is straightforward that placing all nodes in the proxy will achieve maximum security, but in such a way the proxy will be very heavily loaded. The proxies provide the computational services of the software for authorized users. Since one proxy may serve many users who are requesting the delegated execution for privileged applet at the same time, the computation load in the proxy should be an important factor for the overall performance. In such distributed computing environment, the main issue is to allocate as more applets as possible to be executed by user to reduce the load of the proxy server.

Since we also want to keep all applets in the user independent to provide protection, the problem can be reduced to finding the maximum independent set in an arbitrary graph. We define computation cost to be the cost spent in the running time of an applet, including CPU cycles, memory allocated, and disk space used. Since each applet has a different computation cost, the problem becomes finding the maximum weighted independent set in an arbitrary graph.

4.2.1 Finding Maximum Weighted Independent Set

In a graph $G = (V, E)$ and each vertex has a positive weight w. Let S to be the independent set for the graph G if for all $v_i, v_j \in S$, $v_i, v_j \notin E$. The maximum weighted independent set with the weight w for the graph G is to maximize $W(S) = \sum_{i \in S} w_i$. A clique of graph $G = (V, E)$ is the subset $C \subseteq V$, where $G(C)$ is a complete graph. Finding the maximum weighted independent set in G is equivalent to finding the maximum weighted clique in $\overline{G}$, where a maximum weighted clique is a clique that the sum of all of its weighted vertices is maximal.

The problem of finding the maximum weighted or unweighted independent set in an arbitrary undirected graph, has been proven to be NP-hard [Garey79]. The problem is notoriously hard even if vertices of the graph are unweighted. For the unweighted case, an efficient algorithm for finding maximum independent set has been presented by [Tarjan77], which takes $O(E^{1/3})$ time. Many heuristic algorithms have been proposed for finding maximum weighted independent set or maximum weighted clique in an arbitrary graph [Balas96][Kopf87][Pardalos91][Xue94]. Polynomial time algorithms for many other restricted classes of graphs have also been proposed. If the graph is a tree, the maximum weighted independent set can be found in $O(n)$[Chen88].

With the algorithms for finding maximum weighted independent set, the optimal partitioning for the software that the computation load of the proxy is minimum and applets executed by user are independent can be found.

4.2.2 Considering both Computation and Communication Load

Now we consider that the network bandwidth between user and proxy may be limited, and the computing power for the proxy may be also limited. We want to partition the software that gives optimal assignment for load consideration under such limitations. In the applet dependency graph, we define each edge to be the network communication loads between two applets. The communication load is often measured as the average number of messages in an execution session, and it is defined to be zero between to applets if:

1) Two applets are nonadjacent.

2) Two adjacent applets are assigned in the same location.

Then we define communication degree for an applet to be the total communication load between the applet and all other adjacent applets. Here are the steps for calculating the communicate degrees for all applets.

Step 1: Measure the communication loads between any two applets and define as the weights of edges in the graph.

Step 2: Add the weights of all incident edges of a node to be its communication degree, if the adjacent node has not been assigned as the same number with the node.

An example of the steps for calculating the communication degrees for each applet is given in Figure 4.5. In this example, the two darker nodes have been initially assigned as the same number. Therefore the communication load of the edge between them is not added to communication degree for them.

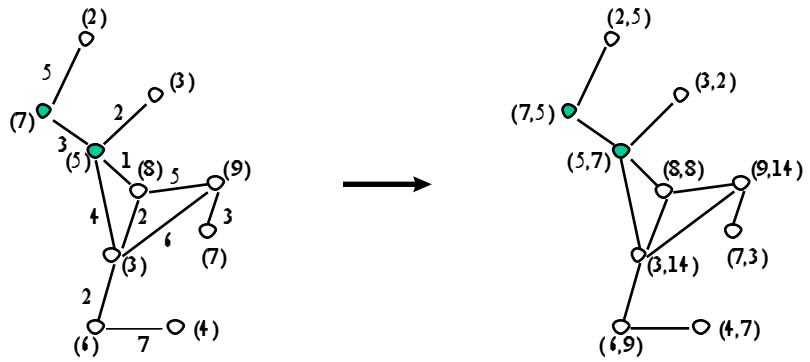


Figure 4.5 Calculating communication degree

Consider that the computing power of proxy or the network bandwidth may be limited. We can formulate our problem for partitioning. First we define some variables that will be used in the problem.

m: computation cost for an applet

M_{all} *: sum of total computation cost for all applets*

n: communication degree for an applet

N_{all} *: sum of total communication degree for all applets*

P: computing power of the proxy

B: the network bandwidth

Our problem for partitioning under different limitations becomes:

1. *Minimize the computation cost under limited network bandwidth between proxy and user*

   Maximize $y = \sum_{i \in S} m_i$ subject to the constraint that $\sum_{i \in S} n_i \le B$, where S is an independent set for graph G.

2. *Minimize the communication load under limited computing power of the proxy*

   Maximize $z = N_{all} - \sum_{i \in S} n_i$ subject to the constraint that $\sum_{i \in S} m_i \le M_{all} - P$, where S is an independent set for graph G.

The independent set S contains the applets that will be executed by the user. The two problems are equivalent, and we formulate our problem as the follows.

Problem

In a graph G=(V,E) where each node has two kinds of weights, defined as (m,n).

Let S to be the independent set for the graph G if for all $v_i Ev_j \in S$, $v_i Ev_j \notin E$. The maximum weighted independent set with the weight *w* for the graph G is to maximize $WESE = \sum_{i \in S} w_i$. The problem is to find the subset S of vertices where $MESE = \sum_{i \in S} m_i$ is maximum under the constraint that $\sum_{i \in S} n_i \le k$, where $k \ge k$. The upper bound of *k* is the maximum weighted independent set for graph G with respect to weight *n*. Therefore, if *k* is assigned as a value greater than or equal to MWIS with respect to weight *n*, the problem becomes finding MWIS with respect to weight *m*.

We have not yet found an efficient algorithm to find the optimal assignment under such constraints. Here we present a simple heuristic method to solve this problem recursively.

A Heuristic Algorithm

Step 1: (1) Set $S = \varnothing EM = kEN = k$.

(2) Set $S_0 = \varnothing EM_0 = kEN_0 = k$.

(3) Set $S_1 = \varnothing EM_1 = kEN_1 = k$.

Step 2: If $G \neq \varnothing$ choose a vertex i in graph G, otherwise stop.

Step 4: For the chosen vertex i, if $n_i > k$ or vertex i has been initially assigned to 1, go to Step 6.

Step 5: Set $G_0 = G - iii - i$ vertices adjacent to ii and $k_0 = k - n_i$. Find

$M_0 \, EN_0 \, ES_0$ by calling the algorithm for graph G_0. If i has been initially assigned to 0, go to Step 7.

Step 6: Set $G_1 = G - iii$ and $k_1 = k$. Find $M_1 \, EN_1 \, ES_1$ by calling the algorithm for graph G_1.

Step 7: If $M_0 > M_1$ then $M \leftarrow M_0 + m_i \, EN \leftarrow N_0 + n_i \, ES \leftarrow S_0 \cup i$.

Otherwise $M \leftarrow M \, EN \leftarrow N_1 \, ES \leftarrow S_1$.

After the steps of the heuristic algorithm, the set S is the applets that will be assigned to the user. Note that in the beginning if $\sum_{i \subset I} n_i > k$ where I is the initially assigned set of independent vertices in G, the process should be stopped because no valid solution with the constraints in graph G can be found.

4.3 Partitioning Between Proxies

In the partitioning model, all applets executed by the user are independent, and the software a user gets consists of minimal independent pieces of applets. Now consider that we want to achieve higher security by minimizing the applets of the software a user can get. We can simply assign all applets to the proxy except the applets that must be executed by user.

We assume that there are many proxies available on the network and each proxy may be compromised. In this case we assign the applets to the proxies by coloring the graph to make each proxy gets independent applets.

The problem of assigning applets to the proxies to make the applets in each proxy are independent can be formulated as the problem of vertex coloring. We discuss the vertex coloring as the follows.

Vertex Coloring

Let G be a graph. A vertex coloring of G assigns colors, usually denoted by 1,2,3,…, to the vertices of G, one color per vertex, so that adjacent vertices are assigned different colors. The minimum number n for which there is an n coloring of the graph G is called the chromatic number of G and is denoted by $\chi(G)$. If $\chi(G)$=k we say that G is k-chromatic.

The problem of coloring vertices in an undirected graph has been shown to be NP complete, i.e., no algorithm has yet been proposed to find the optimal coloring in polynomial time [Aho74]. However there are a number of coloring algorithms, which give approximations to minimal coloring. These heuristic graph coloring algorithms can be used to find good approximations to the chromatic number of those graphs that are too large for the coloring [Clark91]. We will discuss both approximate vertex coloring and exact vertex coloring in the following sections and give the guidelines for partitioning with these algorithms.

4.3.1 Approximate Partitioning

If there are enough proxies available on the network, we can use the approximate coloring algorithms for partitioning, which solve the problem in polynomial time.

In this section, we discuss the coloring algorithms that give approximations to minimal coloring. One of which is the simple sequential algorithm [Welsh67]. The algorithm starts with any ordering of the vertices of the graph G say $v_1, v_2, \ldots, v_n$. Now assign color 1 to v_1. Moving to vertex v_2 color it 1 if it is not adjacent to v_1; otherwise, color it 2. Proceeding to v_3, color it if it is not adjacent to v_1; if it is adjacent to v_1, color it 2 if it is not adjacent to v_2; otherwise color it 3. Proceed in this manner, coloring each vertex with the first available color that has not been used by any of its adjacent vertices.

One of the modifications of the simple sequential algorithm is called smallest-last sequential algorithm [Matula72], which performs better among the similar algorithms by choosing vertices of minimum degree. The smallest-last sequential algorithm uses at most $\max_{x_i \in H} d(x_i) + 1$ colors [Broos41], where $d(x_i)$ is the degree for vertex x_i. In our partitioning for security concerns, these algorithms can not be applied directly because some applets may have been initially assigned to specific proxies. Therefore we proposed a modified smallest-last sequential coloring algorithm to solve the problem for coloring on the graph with some vertices initially assigned for colors.

The Modified Smallest-Last Sequential Coloring Algorithm

Assume that the applets executed by user are assigned as color number 0, and applets executed by proxies are assigned as color number greater than 0 which each color number represents a proxy. In the initial assignment, some applets may have been assigned to designated locations. For the initially assigned proxies, the color numbers are chosen from 1, and increasingly. We first delete the vertices that initially assigned as number 0 and solve the reduced subgraph. The modified smallest-last sequential algorithm is described as follows.

Step 1: (1) Let U be the set of vertices initially assigned as color number 0.

(2) Let P be the set of vertices initially assigned as color numbers greater than 0

(3) Let $H = G - U$, where H is the subgraph of G with all vertices in U deleted

Step 2: (1) List the vertices of P as $x_1, \ldots, x_j$.

(2) Choose x_n to be a vertex of minimum degree in $H - P$.

(3) For $i = n-1, n-2, \ldots, j+1$, choose x_i to be a vertex of minimum degree in the subgraph $H - P - \{x_n, x_{n-1}, \ldots, x_{i+1}\}$.

(4) List the vertices of H as $x_1, \ldots, x_n$.

(5) List the colors available as 1,2,…,n.

Step 3: For each i=1,…,n, if x_i has not been assigned a color, let

$C_i = i$ x HEEuEni, which is the list of colors that could color vertex x_i, otherwise let $C_i = i\ p_i\ i$ where p_i is the initially assigned color for x_i.

Step 4: For each i=1,…,n, if x_j is a vertex adjacent to x_i, and x_j has been Initially assigned as the color p_j, set $C_i = C_i - i\ p_j\ i$.

Step 5: Set i=1.

Step 6: Let c_i be the first color in C_i and assign it to vertex x_i.

Step 7: If x_i has not been initially assigned a color, for each j with $i < j$ and

x_i adjacent to x_j in H set $C_j = C_j - i c_i\ i$.

Step 8: Set $i \leftarrow i + x$ and go to Step 6 if $i \leq n$.

Step 9: For i=1,…,n, c_i is the color assigned for vertex x_i.

After the above steps, the applets can be partitioned such that

1) The user gets minimal information from the applets he has.
2) Applets in each proxy are independent.
3) At most $\max_{x_i \in H} d\,E x_i\ E + x$ proxies are required, where $d\,E x_i\ E$ is the degree for vertex x_i.

4.3.2 Exact Partitioning

In this section, we discuss the exact vertex coloring, which gives partitioning with minimal number of proxies. A graph can be colored optimally by coloring with the first color a maximum independent set M_1 in G, and then coloring with the second color with another maximum independent set M_2 in $G_1 = G - M_1$, and so on until all vertices have been colored. Such kind of coloring algorithms are called optimal independent colorings [Christofides71][Christofides75]. An algorithm based on this kind of colorings which produce good suboptimal solutions with little computational effort is also proposed [Lotfi86].

With the algorithms for maximum independent set discussed earlier, we can partition the software and assign them with minimal number of proxies.

4.3.3 Guidelines for Partitioning between Proxies

Partitioning is easy if there are enough proxies available on the network. The modified smallest-last sequential coloring algorithm proposed earlier can be applied. If the number of color used by the approximate algorithm exceeds the number of proxies, the exact coloring algorithms can be applied. Exact coloring algorithms give the solution to partition with minimal number of proxies. If the number of proxies available is fewer than the chromatic number (minimal number of coloring) for the graph, a ideal partitioning cannot be achieved. In this case, we can use the exact coloring algorithm by assigning an maximum independent M_1 in G to the first proxy, and assign M_2 in $G_1 = G - M_1$ to the second proxy, and so on, until

$n-x$ proxies in n have been used. The remaindering applets (which may not be independent) are assigned to the last proxy. Therefore, applets each proxy are independent, except the last one. And we can concentrate on protecting the last proxy.

If the computational capability for a proxy is limited, more proxies may be needed. Assume that each proxy has its different computational capability, and establishing each proxy requires a different cost. Finding the minimal cost with such constraints important is equivalent to the loading problem discussed in [Elion71].

Chapter 5

Conclusions

The development of Java language created a new environment for software usage. In this environment, dynamically downloaded codes allow users execute any programs they are interested in from the network. To protect the copyright of the software on the Internet, software piracy must be prevented. The growth of network makes this problem more serious. Many approaches have been proposed to prevent software piracy, such as key disks, parallel-port locks, and custom serial-number validations. This scheme with authentication process embedded in the software has been successful in stopping most unauthorized users. However, they cannot effectively protect from been cracked by a smart cracker. Once the software is cracked, it will be then distributed widely on the network.

In this project, a model for software authorization and protection in mobile code systems is proposed. To achieve flexible and global security for the rapid growing network environment, the protection for both the software property and principles in the network environment have been taken into consideration. The privileges to access to these applets are separated and distributed to a number of trusted computational proxies. The execution of software is conducted by cooperation of the applets and the proxies contain them. The user holding part of applets of the software will not be able to use the software without the help of these proxies.

A model for software partitioning in this environment is proposed. Independent applets are assigned to the user, which provide little information without cooperation with applets on the proxies. To increase the performance in this environment, computation load of the proxies and communication load between proxies and users should be minimized. An optimal assignment of applets for the software is also proposed to minimize, under the security considerations, the computation load of proxies and the communication load between proxies and users. To reduce the risk of proxies been attacked, vertex coloring has been applied on the partitioning, which reduces the risk of proxies may been attacked. In the case the intruder can acquire that a proxy is compromised, little information.

References

[Aho74] A. V. Aho, J. E. Hopcroft and J. D. Ullman, "The design and Analysis of Computer Algorithms," pp. 364-404, Addison-Wesley, Reading, MA 1974.

[Balas96] E. Balas and J. Xue, "Weighted and Unweighted Maximum Clique Algorithms with Upper Bounds from Fractional Coloring," Algorithmica 15, pp. 397-412, 1996.

[Barker89] William C. Barker, "Use of Privacy-Enhanced Mail for Software Distribution," Fifth Annual Computer Security Applications Conference, pp. 344-347, 1989.

[Bender96] W. Bender, D. Gruhl, N. Morimoto, and A. Lu, "Techiniques for Data Hiding," IBM System Journal, v. 35, no. 3-4, pp. 313-336, 1996.

[Berghel96] H. Berghel and L. O'Gorman, "Protecting Ownerhip Rights Through Digital Watermarking," IEEE Computer, pp. 101-103, July 1996.

[Best79] R. Best, "Microprossor for Executing Encrypted Programs," US Patent 4, 168396, 1979.

[Bic96] Lubomir F. Bic, Munehiro Fukuda, and Michael B. Dillencourt, "Distributed Computing Using Autonomous Objects," IEEE Computer, August 1996.

[Brooks41] Brooks, R. L. (1941). On Coloring the Nodes of a Network, Proc.Cambridge Philosophical Soc., 37, p. 194.

[Brassil95] J. Brassil, S. Low, N. Maxemchuk, and L. O'Gorman, "Electronic Marking and Identification Techniques to Discourage Document

Copying," IEEE J. Selected Areas in Comm., pp. 1495-1504, Oct. 1995.

[Carzaniga97] Antonio Carzaniga, Gian Pietro Picco, and Giovanni Vigna, "Designing Distributed Applications with a Mobile Code Paradigm," In Proceedings of the 19th International Conference on Software Engineering, Boston, Ma., May 1997.

[Ciancarini97] Paolo Ciancarini and Davide Rossi, "Jada -- Coordination and Communication for Java Agents," In Mobile Object Systems: Towards the Programmable Internet, pages 213-228. Springer-Verlag, April 1997. Lecture Notes in Computer Science No. 1222.

[Chaum90] D. Chaum and H. van Antwerpen, "Undeniable Signatures," Advances in Cryptology-CRYPTO '89 Proceedings, Springer-Verlag, pp. 212-216, 1990.

[Chen88] G. H. Chen, M. T. Kuo, and J. P. Sheu, "An Optimal Time Algorithm for Finding a Maximum Weight Independent Set in a Tree," BIT 28, pp. 353-356, 1988.

[Choudhury94] A. K. Choudhury, N. F. Maxemchuk, S. Paul, and H. G. Schulzrinne, "Copyright Protection for Electronic Publishing Over Computer Networks," Technical Memo BL011382-940428-75TM, AT&T Bell Laboratories, April 1994.

[Christofides71] N. Christofides, "An Algorithm for the Chromatic Number of a Graph," The Computer Journal, 14, p. 38, 1971.

[Christofides75] N. Christofides," Graph Theory," Academic Press, London, 1975.

[Clark91] John Clark and Derek Allan Holton," A First Look at Graph Theory," World Scientific, 1991.

[Curtis94] D. Curtis, "Software Privacy and Copyright Protection," WESCON/94,

Idea/Microelectronics, Conference record, pp. 199-203.

[Dakin95] Karl J. Dakin, "Do You Know What Your License Allows?" IEEE Software, pp. 82-83, May 1995.

[Dean96] D. Dean, E. Felten, and D. Wallach, "Java Security: From HotJava to Netscape and Beyond," Proc. IEEE Symp. Security and Privacy, pp. 190-200, May 1996.

[Donovan94] Stephen Donovan, "Patent, Copyright and Trade Secret Protection for Software," IEEE Potentials, pp. 20-24, August/September 1994.

[Elion71] S. Elion and N. Christofides, "The Loading Problem," Manage. Sci. 17, pp. 259-268, 1971.

[Garey79] M. R. Garey and D. S. Johnson, "Computers and Intractability: A guide to the Theory of NP-Completeness," Freeman, San Francisco, CA., 1979.

[Ghezzi97] Carlo Ghezzi and Giovanni Vigna, "Mobile Code Paradigms and Technologies: A Case Study," In Proceedings of the First International Workshop on Mobile Agents, Berlin, Germany, April 1997.

[Goldreich96] O. Goldreich and R. Ostrovsky, "Software Protection and Simulation on Oblivious RAMs," Journal of the ACM , Vol. 43, No. 3, pp. 431-473, May 1996.

[Gong97] L. Gong, "New Security Architectural Directions for Java (Extended Abstract)" . In Proceedings of IEEE COMPCON, San Jose, California, pp. 97-102, Feb. 1997.

[Gosling96] J. Gosling and H. McGilton, "The Java Language Environment," Sun Microsystems, May 1996. http://java.sun.com/doc/language_environment/.

[Gray95] Robert S. Gray, "Agent Tcl: A Transportable Agent System," In

Proceedings of the CIKM Workshop on Intelligent Information Agents, Baltimore, Md., December 1995.

[Hall96] David Hall, Jean Bacon, and John Bates, "Flexible Distributed Programming Using Mobile Code," In Proceedings of the Seventh ACM SIGOPS European Workshop, Connemara, Ireland, September 1996.

[Harn92] L. Harn, H.Y. Lin and S. Yang, "A Software Authentication System for Information Integrity," computers and Security, Vol.11, No.4, pp. 747-752, 1992.

[Herzberg86] A. Herzberg and S. Pinter, "Public Protection of Software," Advances in Cryptology:Proc. Crypto 85, H. C. Williams, Ed., pp. 158-179, 1986.

[Jaeger96] Trent Jaeger, Aviel D. Rubin, and Atul Prakash, "Building Systems that Flexibly Control Downloaded Executable Content," In Proceedings of the 6th Usenix Security Symposium, pages 131-148, San Jose, Ca., July 1996.

[Kahan95] J. Kahan, "A Distributed Authorization Model for WWW," Proc. INET'95 Conf., Honolulu, Hawaii, http://www.isoc.org/HMP/PAPER/107, 1995.

[Kent80] S. T. Kent, "Protecting Externally Supplied Software in Small Computers," Ph.D. dissertation, MIT/LCS/TR-255. MIT, Cambridge, Mass, 1980.

[Lotfi86] Vahid Lotfi and Sanjiv Sarin, "A Graph Coloring Algorithm for Large Scale Scheduling Problems," Comput. & Ops. Res. Vol. 13, No. 1, pp. 27-32, 1986.

[Kopf87] R. Kopf and G. Ruhe, "A Computational Study of the Weighted

Independent Set Problem for General Graphs," Foundations of Control Engineering, pp. 167-180, 1987.

[Matula72] D.W.Matula, G.Marble, and J.D.Isaacson, "Graph coloring algorithms," In R.C. Read, editor, Graph Theory and Computing, pp. 109-122, 1972.

[Neff94] Richard E. Neff, "Software Piracy: International Copyright Overview," WESCON/94, Idea/Microelectronics, Conference record, pp. 190-195.

[Nog96] Saurab Nog, Sumit Chawla, and David Kotz, "An RPC Mechanism for Transportable Agents," Technical Report TR96-280, Department of Computer Science, Dartmouth College, Hanover, N.H., 1996.

[Pardalos91] P. M. Pardalos and N. Desai, "An Algorithm for Finding a Maximum Weighted Independent Set in an Arbitrary Graph," Int. J. Comput. Math. 38, pp. 163-175, 1991.

[Perret96] Stephane Perret and Andrzej Duda, "MAP: Mobile Assistant Programming for Large Scale Communication Networks," In Proceedings of the IEEE International Conference on Communications, Dallas, Tex., June 1996.

[Rubin95] Aviel D. Rubin, "Trusted Distribution of Software Over the Internet," Proc. IEEE Symp. On Network and Distributed System Security , pp. 47-53, 1995.

[Samarati96] P. Smarati., E. Bertino, and S. Jajodia, "An Authorization Model for a Distributed Hytertext System," IEEE Transactions on Knowledge and Data Engineering, Vol 8, No. 4, pp. 555-562, August 1996.

[Sun96a] "Remote Method Invocation Specification", Sun Microsystems Inc. http://www.javasoft.com/products/jdk/1.1/docs/guide/rmi/spec/rmiTOC.doc.html.

[Sun96b] "Signed Applets and Digital Signatures," Sun Microsystems Inc. http://java.sun.com/products/JDK/1.1/docs/guide/signing.

[Tarjan77] R. E. Tarjan and A. E. Trojanowski, "Finding a maximum independent set," SIAM J. Comput., 6, no. 3, pp. 537-546, 1977.

[Voelker86] J. Voelker and P. Wallich, " How Disks are 'Padlocked'," IEEE Spectrum, p. 32, June 1986.

[Wallach97] Dan S. Wallach, Dirk Balfanz, Drew Dean, and Edward W. Felten. Extensible security architectures for Java. Technical Report 546-97, Department of Computer Science, Princeton University, April 1997.

[Welsh67] D.J.A. Welsh and M.B. Powell, "An Upper bound for the Chromatic Number of a Graph and its Application to Timetabling Problems," Comput. J., 10:85-86, 1967.

[White90] S. R. White and L. Comerford, "ABYSS: Architecture for Software Protection," IEEE Transactions on Software Engineering, Vol. 16, No. 6, pp. 619-629, June 1990.

[Wilson97] A. Wilson, "Software Security and the DirectPlay API," Dr. Dobb's Journal, p. 66, April 1997.

[Xue94] J. Xue, "Edge-Maximal Triangulated Subgraphs and Heuristics for Maximum Clique Problem," Networks, Vol. 24, pp. 109-120, 1994.