ELSEVIER

Computer Communications 22 (1999) 46–55

computer communications

Optimal assignment of mobile agents for software authorization and protection ☆

Shiuh-Pyng Shieh*, Chern-Tang Lin, Shianyow Wu

Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu 30010, Taiwan

Received 10 November 1997; accepted 21 October 1998

Abstract

In this paper, a model for software authorization and protection in mobile code systems is proposed. In the model, a software is partitioned into objects, called mobile agents, and the privileges to access these agents are separated and distributed to the user's local system and a number of trusted servers called trusted computational proxies. The execution of a program (software) is conducted by cooperation of the agents and the proxies that contain them. Two agents are dependent if there is a message passing between them. To reduce the risk of software being attacked, dependent agents are distributed to different proxies. In this way, if a proxy is compromized, minimal information of the software will be disclosed. Methods for assigning agents to proxies are also proposed to minimize, under the security constraints, computation load of the proxies as well as communication load between the user's local system and proxies.

Keywords: Software protection; Mobile code; Remote execution; Java language; Proxy

1. Introduction

The rapid development of network and advanced technologies enable new software capabilities and wide market interest, but software piracy, such as the unauthorized copying, use, or distribution of software products, is still a serious and tough problem to cope with. Although various software protection schemes, have been proposed, software piracy still causes major losses to software vendors since some protection schemes can be easily cracked by a malicious user and some require additional costs for users [12,13,15,25,32,35].

Most of these software protection schemes embed access control mechanisms in the program code, and a user has to pass these authentication processes before using the software. The process may require serial number of the corresponding user, password from the manual, or checking the source where the software locates (CD or floppy disk, for example). Unfortunately, these authentication processes have been cracked by many crackers (Fig. 1). The difficulty of cracking such a protection scheme depends on how complex 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 later and the execution will be terminated. This just increases the time to crack the software, however, it cannot prevent unauthorized use.

Recent advance of network technology allows network users to access the Internet in a more effective way. The growing importance of Internet has stimulated research on a new generation of programming languages. Recently, mobile code languages [17,19,20] have been proposed as a technological answer to the problem. These languages view the network and its resources as a global environment in which computations take place [5–7]. A mobile-code-based software is partitioned into objects, called mobile agents. For example, in mid-1995, Sun Microsystems announced the Java language [19]. 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 agents, called applets in Java, and running these agents locally [28].

The development of mobile code technologies changes the style of software usage. The mobility and cross-platform characteristics of mobile agents allow software rental on the network. Users can download the corresponding agent of

☆ This work was supported in part by the National Science Council, Taiwan under contract NSC 86-2622-E-009-007R.

* Corresponding author. Tel.: 886 3573 1876; fax: 886 3572 4176; e-mail: ssp@csie.nctu.edu.tw

0140-3664/99/$ - see front matter
PII: S0140-3664(98)00254-0

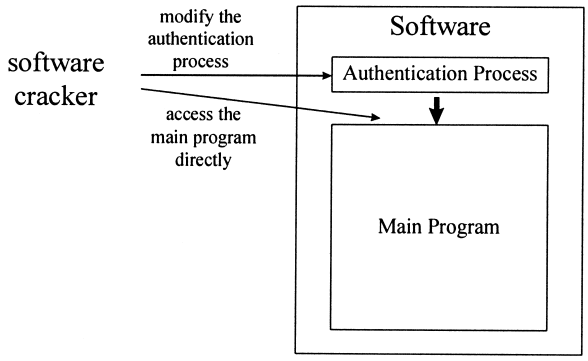

Fig. 1. Common software protection schemes.

software across the network and run it dynamically when they want to execute some functions of the software. They will no longer be asked to purchase the entire software when they just need to use part of the features. Revision for software in the environment becomes simple. Contrarily, software developers can always provide the newest software for users, and can know how many times a program has been downloaded by a user. However, illicit dissemination of software appears to be more serious on the network. It is desirable to control the access that only authorized users can download and execute a program. When a user wants to download a program from the service provider, the conditional access can be achieved by appropriately setting download permissions. But the service provider has no control over the mobile agents that have been distributed to users. That is, the new style of software usage on the network causes more serious software-piracy problem, and, similarly, common software protection schemes that relies on the authentication process within the software itself cannot effectively prevent the software from being cracked by a smart cracker.

To deal with the problem of software piracy on the network, not only the software itself but also the environment associated with the software must be considered. The compromised version of software may be harmful to users executing it, since it may contain a Trojan Horse or virus [3,14,27]. The malicious code that contains a Trojan Horse or virus accessing user's system resources such as the file system, the CPU, the network, and the graphics display may cause unpredictable effects, such as stealing user's privacy or damaging resources in user 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. Therefore, not only mobile-code-based softwares require a good software authorization and protection model to prevent software piracy, but also users need a secure environment against the attacks from malicious mobile agents.

To distribute the software in a secure manner that prevents users' local systems from attacks of maliciously modified agents, digital signature can be applied. Many code distribution mechanisms have been proposed to enforce trusted distribution of software [3,21,27,37]. In JDK 1.1 (Java Development Toolkit) [18,29], the code signing feature is provided and the user who downloads the agent can identifying the sender by verifying the signature. If the agent is not trusted, execution will be restricted in a sandbox with only limited system resource provided.

Another Java-based mobile agent, called aglet, was developed at IBM's Tokyo Research Laboratory [31]. Aglets are able to automatically visit aglet-enable hosts, execute on them, and communicate with other aglets in the computer network. Like other mobile agents, aglets are a potential threat to a system and they are also exposed to threats by their hosting system. Karjoth et al. thus proposed a security model for the aglets development environment [22]. But, like other literatures which discuss the security issues of mobile agents, their security model currently only focus on protection of the host against aglets. That is, the application (or software) composed of aglets will suffer from the problems of software piracy from malicious hosts (users), such as unauthorized use or illicit dissemination.

In this paper, we will propose a software authorization and protection model which emphasizes the protection for mobile-code-based software (or the software vendors) to prevent the attacks of hosts (users). To achieve flexible and global security for the rapid growing network environment, the protection of the software property in the network environment has been taken into consideration. In the model, the privileges to access the agents of a program are separated and distributed to the user's local system and a number of trusted servers called trusted computational proxies. Dependent agents are distributed to different proxies to minimize the information disclosure in case a proxy is compromised. In the environment, methods for assigning agents are also proposed to minimize, under the security constraint, computation load of the proxies as well as communication load between the user and proxies.

This paper is organized as follows: in Section 2, our proposed model for software authorization and protection is presented, which is based on the concept of separation of execution privileges. In Section 3, a model for software partitioning to achieve protection in this environment is presented, and related issues for achieving better performance and security will be discussed. Finally, we give the conclusions in Section 4.

2. The proposed authorization and protection model

In mobile code systems, a program (software) is composed of a number of agents. An agent can be downloaded dynamically from the remote machine and executed on the local machine, and a job can be processed by the

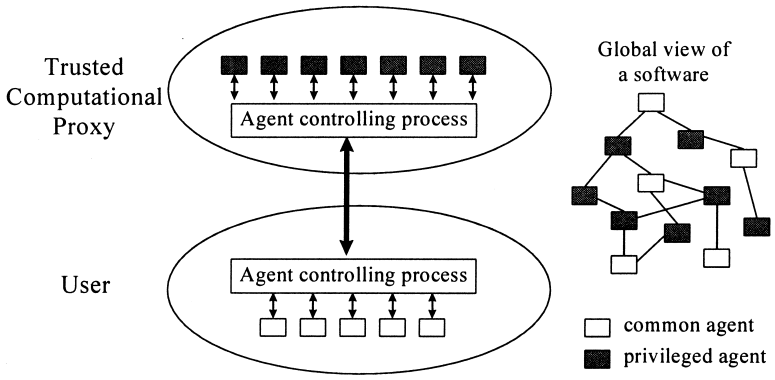

Fig. 2. The proposed protection model.

cooperation of these agents. In the section, an authorization and protection model is proposed to enhance the security and protection of mobile codes by delegating some critical execution services to one or more trusted and protected proxies.

2.1. System model overview

The execution of a program can be considered to include three parts: incoming messages, transformation processes, and outgoing messages. An agent participates in the transformation process for a message if the agent sends or receives the message. If some critical agents are removed from a program, execution of the program cannot proceed.

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

The trusted computational proxy provides computation services for privileged agents, as shown in Fig. 2. Only a trusted proxy has the capability to get privileged agents and execute them. The proxy executes the agents on behalf of an authorized user and returns the result to the user. In this way, an unauthorized user cannot acquire the results of privileged agents, and therefore benefit little from the software. A program may consist of many proxies, and each proxy executes only a subset of the privileged agents. Thus, a compromised proxy will not leak all privileged agents. In the proposed model, agents to be downloaded are encrypted by agent keys, and the agent keys for each agent are different. These keys are only available to trusted proxies or authorized users. The model consists of six major components:

Software Vendor
The company who developed the software.
Certification Authority
The party who signs and issues the certificate containing the user's public key.
Software Authentication Center
An accredited organization that authenticates the software developed by software vendors, and signing legitimate parts of the software.
Agent Server
The server who stores agents provided by software vendors. When a host wants to execute an agent, it first downloads the agent from an agent server.
Trusted Computational Proxy
The server that provides computational services of privileged agents for users.
User The user who uses the software.

2.2. 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 agent and the execution license gives the right for user or proxy to download and execute an agent. Note that this paper does not clearly describe the detailed format of licenses since the implementation issue is out of the scope of the paper.

2.2.1. Publication license

In our environment, secure distribution of agents is achieved by the publication licenses. For each agent, there is a publication license associated with it. The publication license is issued and signed by software authentication center, and every agent provided by a software vendor must have a legal publication license.

A publication license consists of:

1. Serial number,
2. Software vendor information,
3. Software authentication center information,
4. Agent information (ID, version),

5. Message digest of the agent, and
6. Issuing and expiration time.

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

When the software vendor releases a new agent, it first sends it and the related information about the agent (for example, specification or source code) to the software authentication center. The software authentication center checks the agent, and if there is no problem with it, the center issues a publication license of this agent and sends back to software vendor.

2.2.2. Execution license

The user or proxy must get an execution license to execute the corresponding agent. The execution license is issued and signed by software vendor to prevent an unauthorized user from forging it.

The execution license consists of:

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

The execution capability of an agent determines whether a user or a proxy can download the agent. If the user has the execution capability of an agent, he can get some extra information of the agent from software vendor, for example, agent key or message digest of the agent, to decrypt and verify the agent. To execute the privileged agents that have to be executed in the proxies, a user must have the delegation capability for these agents. That is, the delegation capability determines whether a user can delegate the execution of an agent, which he cannot execute directly, to the proxy.

The execution and delegation capabilities for a user depends on how many agents 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 agents providing these features.

The rest of the license contains the other basic information, such as the user's certificate identifying who will execute the agent, the information of the software vendor who produced the agent, the expiration time of this license, and so on. The information helps the agent servers to verify the legality of the execution licenses while the user/proxy requests to download agents.

2.3. Using the software

The user can purchase the execution license of the software he is interested in from the software vendor. Once the user receives the execution license issued 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.

2.3.1. Agent downloading

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

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

2.3.2. Execution of the software

After a user downloads the agents from an agent server, he can begin to execute them. Since privileged agents are forced to be executed by the proxies, the user has to bind these agents first before execution. In the binding phase, the user sends his execution license to the proxy server he wants the execution to be delegated. The user and the proxy mutually authenticate execution license of the other. The execution then proceeds by executing the agents corresponding to the capabilities listed in user's execution license.

If there are more than one proxy participating in the execution, the user will be required to explicitly make connections to each of them and authenticate with each other. For using the software for the first time, the user requests the software vendor for a list of available proxies. He then chooses the proxy for computational service and registers himself at this proxy. Registration for execution licenses will be discussed in the next section.

2.4. 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 of a user if illegal behavior of the user has been found. Moreover, with the registration and revocation support, it is desirable to record in a license the number of executions granted to a user. The proxy records the number of executions invoked by the user, and if it exceeds the limitation recorded in the user's execution license, subsequent execution will be rejected.

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 corre-

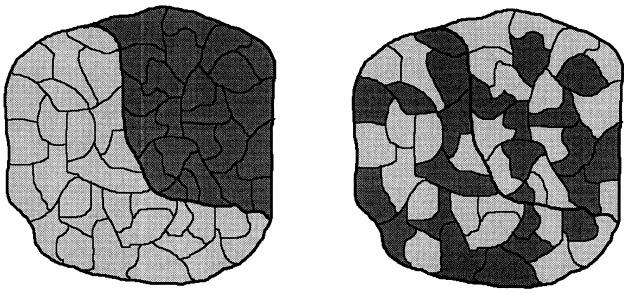

Fig. 3. Example of partitioning.

sponding registry $[U, P, SN_U]_{D_s}$ (signed by the software vendor, where U is the user's identity, P the proxy's identity, D_s the software vendor's private key, and SN_U the serial number of the license) 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 invalid. The registration steps are described as follows:

Step 1 A user sends a request along with his execution license to the software vendor for registration at a proxy.
Step 2 The software vendor checks validity of the user's license. Go to Step 3 if valid, otherwise stop.
Step 3 The software vendor sends a message $[U, P, SN_U]_{D_s}$ (signed by the software vendor) to the new proxy to add user's record at the proxy.
Step 4 The software vendor updates its own registry for the user.

To revoke an execution license of a user in a proxy, the software vendor simply tells the proxy to remove the registry for the user and then removes the registry located at the software vendor itself. Then the user's execution license will be revoked because no registries can be found in the proxy. In addition, the vendor can also inform the user that the license is revoked.

3. Software partitioning

Software partitioning means separating agents such that a user cannot benefit from holding only a subset of agents of a program. The goal is to partition the software in such a way that a user holding a single agent or a subset of the agents will not be able to get an acceptable result if acquiring the result requires the help of other agents.

In Fig. 3, we compare two different ways of software partitioning. Assume that each area in the graph represents a code fragment, that is, an agent. The execution of an agent depends on the execution of adjacent agents, that is, there will be message passing between the adjacent code fragments. There are two partitions in the graph, where a light-color area represents an agent to be executed by a user, and a dark-color area represents an agent to be executed by a trusted computational proxy. It is clear that the partitioning on the right of Fig. 3 provides better protection than the partitioning on the left. The method of partitioning in the left graph simply cut the program into two halves, where the left half will be given to the user and the right half will be given to the proxy. If an authorized user acquires any half of the program, he can still execute partial functions of the software and get some results. Contrarily, the method on the right divides the program into small pieces, and distributes them to the user and proxy server. In case either one of them is compromised, an intruder can benefit little from the compromised agents, because many of the agents he received relies on execution of the other agents.

The execution of an agent may disclose some information to the user. The more agents the user can get, the more information may be gained from the user. If the user gets all the agents, we can say that the whole software is compromised. However, for two nonadjacent compromised agents, since they are not directly dependent, the intruder can only acquire two small pieces of information from them, but cannot find the relationship between the two pieces. However, for two adjacent agents, the intruder can find their relationship and merge the two pieces of information to acquire more information.

3.1. Proposed partitioning model

A program in the mobile code system can be represented as an undirected dependency graph $G = (V, E)$, where a vertex represents an agent and an edge represents the dependency between two agents. If an agent may communicate with another agent, they are dependent. For two dependent agents in execution, there will be messages passing between them.

In the software, we assume that the user can get more acceptable result from it if he can get a larger subset of the connected agents. Giving the user two independent agents will provide better protection than two dependent agents, because the user cannot benefit from two independent agents directly if they depend on other agents executed in the proxy. Based on the assumption, we proposed a

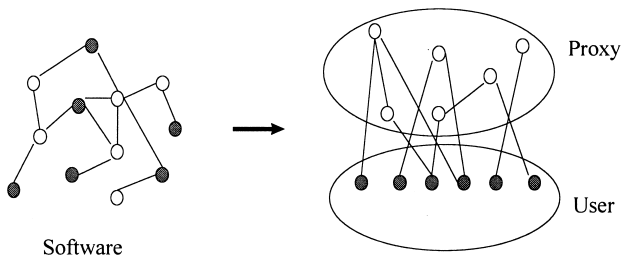

Fig. 4. The proposed partitioning model.

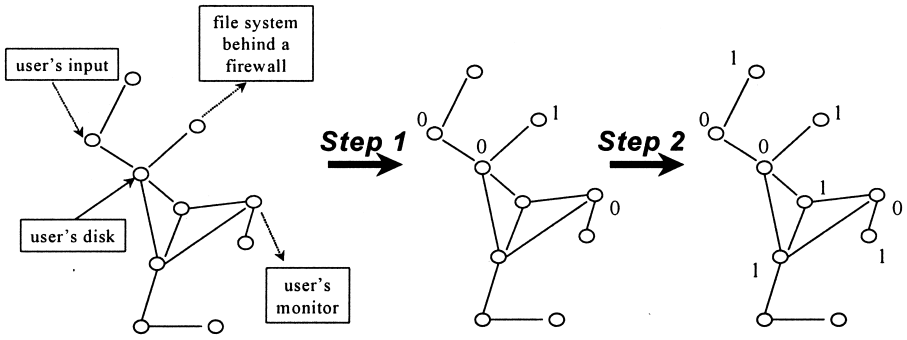

Fig. 5. An example for initial agent assignment.

partitioning model, in which any two agents executed by the user are independent, as shown in Fig. 4.

In the scheme, each agent on the user's machine depends on the agents executed by the proxy. The partitioning model considers the security only at the agent level, and the internal structure of an agent will not be covered in this paper. An agent is a basic element in the model. For a small software with only several agents, a heuristic partitioning may work well. However, for a large software composed of many agents, our model gives a good protection by partitioning the software into pieces which will be assigned to different participants to acquire better security. In the assignment, we consider the following two issues:

1. Performance and
2. software protection.

For the first issue, we wish to achieve good performance by reducing the computational load of the proxy and distributing more agents to the user. Since the proxy provides the computation services for many users, its load is usually rather heavy and it may become a bottleneck. The computation load on the proxy should be an important factor for the overall performance. To reduce the computation load on the computation proxy, it is desirable to distribute as many agents to the user as possible. Contrarily, for the second issue, it is desirable to distribute as few agents to the user as possible to reduce the possibility of software piracy. To balance the two requirements, a possible approach is to assign as many agents to the user as possible under the constraint that all agents executed by the user are independent. Further, if each agent has a different computational cost, it is also desirable to find an assignment that minimizes the computation load on the proxy. In Section 3.3, we will also take the communication load between any two agents into consideration, and find the optimal assignment for reducing both computation cost and communication load.

3.2. Assignment of agents

In this section, we will discuss the method for assigning agents to participants, including the user and the proxy. In the dependency graph for a program, the agents assigned to be executed by the user is marked number 0, and those executed by the proxy are marked number 1 or greater. Not all agents of the program will be freely assigned. Some agents may have special properties and have to be assigned at specific locations. Before partitioning, we find these kind of agents and assigned them first. The steps for the initial assignment are described as follows, and an example is shown in Fig. 5.

Step 1: Mark the nodes that have to be placed at specific locations
Some agents have to be executed at specific locations. For example, some agents may be designed for reading data from the user's keyboard, displaying data to the user's monitor, or reading/writing data from the user's hard disk. These agents have to be executed by the user, and marked number 0. Some agents have to open some network connections from a proxy (in a firewall, for example) or reading/writing something from the proxy's file system. These agents should be placed in the proxy, and marked number 1. In this step, all special nodes (agents) are marked a number depending on the location the agents has to be assigned.
Step 2: The nodes adjacent to nodes with 0 are marked 1
Since the agents executed by the user must be independent, all agents adjacent to agents with number 0 cannot be marked number 0 again. These agents have to be marked number 1.

Here are some examples for these kind of agents that have to be initially assigned.

1. User
   - Reading from keyboard,
   - Reading or writing from user's hard disk,
   - Displaying on the monitor, and
   - Communicating with network with user's identity.
2. Proxy
   - Reading or writing from proxy's file systems,
   - Execution from behind the firewall, and
   - Agents consisting of critical codes.

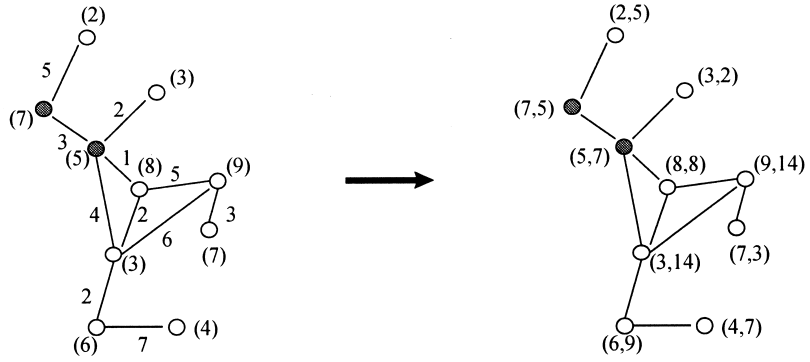

Fig. 6. Calculating communication degree.

In addition, the security concerns is also an important factor for the initial assignment of agents. In the network environment, sometimes an agent may invoke some operations on a specific principle, and the correct execution must be assured. For example, the software vendor may want to record execution states of the software provided for users. Sometimes a database can be only accessed by a trusted party. If the execution of the agent is performed by the user, he may modify the code to deviate from prescribed execution and creates faulty results. Thus these agents have to be assigned to the proxy to ensure correct results.

3.3. Partitioning for performance considerations

If the proxies are trusted and protected, the assignment of adjacent agents to the same proxy will not be a problem. Since the unauthorized users cannot access agents in the proxies, we just need to assign the nodes in such a way that all agents in the user are independent. The assignment problem is thus reduced to finding the maximum independent set in a dependency graph. Since each agent usually need different execution time, we assign a weight to each agent, where an agent with heavy weight imposes more computation cost than an agent with light weight. Since each agent has a different computation cost, the assignment problem is reduced to finding the maximum weighted independent set in an arbitrary graph.

3.3.1. Finding maximum weighted independent set

In a graph $G = (V, E)$, and each vertex v_i has a positive weight w_i. Let S to be the independent set for the graph G if for all $v_i, v_j \in S, \overline{V_i V_j} \notin E$ The maximum weighted independent set for the graph G is to maximize $W(S) = \sum_{v_i \in S} w_i$. 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 $\bar{G}$, 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 [16]. 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 [30], which takes $O(2^{n/3})$ times. Many heuristic algorithms have been proposed for finding maximum weighted independent set or maximum weighted clique in an arbitrary graph [2,24,26,36]. 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 $O(n)$ [8].

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 agents executed by user are independent can be found.

3.3.2. Considering both computation and communication load

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

1. the two agents are nonadjacent, or
2. the two agents are adjacent but assigned to the same location.

Then we define communication degree of an agent to be the total communication load between the agent and all other adjacent agents. Here are the steps for calculating the communication degree of an agent.

Step 1 Measure the communication load between any two agents and define it as weight of the edge 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 marked the same number as the node.

An example of the procedure for calculating the communication degrees of each agent is given in Fig. 6. In the graph preceding the arrow, each agent is labeled its computation load and each edge is labeled its communication load. In the graph following the arrow, each vertex (agent) v_i is labeled the pair (m_i,n_i), where m_i represents computation load of the agent, and n_i represents the communication degree of the agent. Since the two dark agents have been initially marked the same number and assigned to the same host, communication load of the edge between them is not added to communication degrees of both agents.

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

m_i computation cost of agentv_i,
M_{all} total computation cost of all agents,
n_i communication degree of agentv_i,
N_{all} total communication degree of all agents,
P computing power of the proxy, and
B the network bandwidth.

The problem for partitioning under different limitations becomes:

Minimize the computation cost under limited network bandwidth between the proxy and user
Maximize $y = \sum_{v_i \in S} m_i$ subject to the constraint that$\sum_{v_i \in S} n_i \leq B$, where S is an independent set for graph G.
Minimize the communication load under limited computing power of the proxy
Maximize $z = N_{all} - \sum_{v_i \in S} n_i$ subject to the constraint that $\sum_{v_i \in S} m_i \leq M_{all} - P$, where S is an independent set for graph G.

The independent set S contains the agents that will be executed by the user. The two problems are equivalent, and we formulate our problem as follows. According to the description of Section 3.3.1, the problem is to find the subset S of vertices such that $M(S) = \sum_{v_i \in S} m_i$ is maximum under the constraint $\sum_{v_i \in S} n_i \leq k$, where$k$ is a given upper bound and $0 \leq k \leq B$. Here we present a heuristic method to solve this problem recursively. The algorithm is able to find, under a given network bandwidth constraint, the independent set with maximum computation weight for graph G.

3.3.2.1. Algorithm for finding the independent set with maximum computation weight

Step 1 (1) Set $S = \emptyset$, $M = 0$, $N = 0$; (2) Set $S_0 = \emptyset$, $M_0 = 0$, $N_0 = 0$; and (3) Set $S_1 = \emptyset$, $M_1 = 0$, $N_1 = 0$.
Step 2 If $G \neq \emptyset$ choose a vertex i in graph G, otherwise stop.
Step 3 For the chosen vertex v_i, if $n_i > k$ or vertex v_i is initially assigned 1, go to Step 5.
Step 4 Set $G_0 = G - \{v_i\} - \{$vertices adjacent to $v_i\}$ and $k_0 = k - n_i$. Find M_0, N_0, S_0 by calling the algorithm for graph G_0. If vertex v_i is initially assigned 0, go to Step 6.
Step 5 Set $G_1 = G - \{v_i\}$ and Find M_1, N_1, S_1 by calling the algorithm for graph G_1.
Step 6 If $M_0 > M_1$
then$M \leftarrow M_0 + m_i, N \leftarrow N_0 + n_i, S \leftarrow S_0 \cup v_i$.
Otherwise $M \leftarrow M, N \leftarrow N_1, S \leftarrow S_i$.

After executing the heuristic algorithm, the set S consists of the agents to be assigned to the user. Note that in the beginning if $\sum_{v_i \subset I} n_i > k\}$ where I is the initially assigned set of independent vertices in G, the process should be stopped because no valid solution satisfying the constraint in graph G can be found. Further, if k is large enough to support all possible communication between the user and proxy, the partitioning problem is reduced to finding the independent set with maximum computation weight.

3.4. Partitioning agents among multiple proxies

In the previous section, we investigate the methods for partitioning agents between a user and a proxy. In the section, we will investigate the partitioning method for the network environment with multiple proxies, where each proxy may be compromised. To reduce the risk of software piracy, we assign the agents to the proxies in the way that each proxy gets independent agents. Thus, the disclosure of software information can be minimized. The problem of assigning independent agents to each proxy can be formulated as the vertex coloring problem. We discuss the vertex coloring as follows.

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 [1]. 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 [11]. We will discuss both approximate vertex coloring and exact vertex coloring in the following sections and give the guidelines for partitioning with these algorithms.

3.4.1. 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

approximation to minimal coloring. One of the coloring algorithm is the simple sequential algorithm [33]. The algorithm starts with any ordering of the vertices of the graph G, say $V_1,\ldots,V_n$. It first assigns color 1 toV_1; then moves to vertex V_2 and colors it 1 if it is not adjacent to V_1; otherwise, colors it 2. Proceeding toV_3, color it 1 if it is not adjacent toV_1; color it 2 if it is adjacent toV_1, but not adjacent toV_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. In the following, we proposed a new smallest-last sequential assigning algorithm to solve the assigning problem with some vertices initially assigned.

3.4.1.1. The smallest-last sequential assigning algorithm Assume that the agents executed by user are assigned color number 0, and agents executed by proxies are assigned color number greater than 0 which each color number represents a proxy. In the initial assignment, some agents 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 number 0 and solve the reduced subgraph. The smallest-last sequential assigning algorithm is described as follows:

1. Step 1
   - Let U be the set of vertices initially assigned color number 0.
   - Let P be the set of vertices initially assigned color numbers greater than 0.
   - Let $H = G - U$, where H is the subgraph of G with all vertices in U deleted.
2. Step 2
   - List the vertices of P as $x_1,\ldots, x_a$.
   - Choose x_n to be a vertex of minimum degree in $H - P$.
   - For $i = n - 1, n - 2,\ldots,a + 1$, choose X_i to be a vertex of minimum degree in the subgraph $H - P - \{x_n, x_{n-1,\ldots,x_{a+1}}\}$.
   - List the vertices of H as $x_1,\ldots,x_n$.
   - List the colors available as $1,2,\ldots,r$.
3. Step 3
   - For all x_i, $i = 1,\ldots,a$, let $C_i = \{P_i\}$ where P_i is the initially assigned color for x_i.
   - For all x_i, $i = a + 1,\ldots,n$, let $C_i = \{1,2,\ldots,r\}$, which is the list of colors that can color vertex x_i.
4. Step 4
   - Set $i = 1$.
5. Step 5
   - If $i > a$, let C_i be the first color in C_i and assign it to vertex x_i.
6. Step 6
   - Set $C_j = C_j - \{c_i\}$ for each X_j in H, $j > i$, and x_j adjacent to x_i.
7. Step 7
   - Set i larr; $i + 1$ and go to Step 5 if $i \leq n$.
8. Step 8
   - For $i = 1,\ldots,n$, c_i is the color assigned to vertex x_i.

After executing the algorithm, the agents can be partitioned such that

1. The user gets an independent set of agents.
2. Agents in each proxy are independent.
3. At most $\max_{x_i \in H}[d(x_i)] + 1$ proxies are required, where $d(x_i)$ is the degree for vertex x_i.

3.4.2. Optimal 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 [9,10]. With the algorithms for maximum independent set discussed earlier, we can partition the software and assign them with minimal number of proxies.

3.5. Guideline for partitioning among proxies

Partitioning is easier if there are enough proxies available on the network. The smallest-last sequential assigning algorithm proposed earlier can be applied. If the number of colors 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, an 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 $n - 1$ proxies in n have been used. The remaining agents (which may not be independent) are assigned to the last proxy. Therefore, agents on each proxy are independent, except the last one. And we can concentrate on protecting the last proxy.

4. Conclusions

In this paper, 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. In the proposed model, a software consists of agents. The privileges to access these agents are separated and distributed to a number of trusted computational proxies. The execution of a software are conducted by cooperation of the agents and the proxies containing them. The user holding part of agents of the software will not be able to use the software without the help of these proxies.

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

For further reading

See Refs. [4,23,34].

References

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

[2] E. Balas, J. Xue, Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring, Algorithmica 15 (1996) 397–412.

[3] W.C. Barker, Use of privacy-enhanced mail for software distribution, Fifth Annual Computer Security Applications Conference, 1989, pp. 344–347.

[4] R. Best, Microprossor for executing encrypted programs, US Patent 4, 168396, 1979.

[5] L.F. Bic, M. Fukuda, M.B. Dillencourt, Distributed computing using autonomous objects, IEEE Computer August (1996).

[6] A. Carzaniga, G.P. Picco, G. Vigna, Designing distributed applications with a mobile code paradigm, in: Proceedings of the 19th International Conference on Software Engineering, Boston, MA, May 1997.

[7] P. Ciancarini, D. Rossi, Jada – coordination and communication for Java agents, in: Mobile Object Systems: Towards the Programmable Internet, Lecture Notes in Computer Science No. 1222, Springer-Verlag, Berlin, 1997, pp. 213–228.

[8] G.H. Chen, M.T. Kuo, J.P. Sheu, An optimal time algorithm for finding a maximum weight independent set in a tree, BIT 28 (1988) 353–356.

[9] N. Christofides, An algorithm for the chromatic number of a graph, The Computer Journal 14 (1971) 1971.

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

[11] J. Clark, D.A. Holton, A First Look at Graph Theory, World Scientific, Singapore, 1991.

[12] D. Curtis, Software Privacy and Copyright Protection, WESCON/94, Idea/Microelectronics, Conference record, pp. 199–203.

[13] K.J. Dakin, Do you know what your license allows?, IEEE Software (1995) 82–83.

[14] D. Dean, E. Felten, D. Wallach, Java security: from HotJava to Netscape and beyond, Proc. IEEE Symp. Security and Privacy, May 1996, pp. 190–200.

[15] S. Donovan, Patent, copyright and trade secret protection for software, IEEE Potentials August/September (1994) 20–24.

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

[17] C. Ghezzi, G. Vigna, Mobile code paradigms and technologies: a case study, in: Proceedings of the First International Workshop on Mobile Agents, Berlin, Germany, April 1997.

[18] L. Gong, New security architectural directions for Java (extended abstract), in: Proceedings of IEEE COMPCON, San Jose, California, February 1997, 97–102.

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

[20] R.S. Gray, Agent Tcl: a transportable agent system, in: Proceedings of the CIKM Workshop on Intelligent Information Agents, Baltimore, MD, December 1995.

[21] L. Harn, H.Y. Lin, S. Yang, A software authentication system for information integrity, Computers and Security 11 (4) (1992) 747–752.

[22] G. Karjoth, D.B. Lange, M. Oshima, A security model for aglets, IEEE Internet Computing, 1997.

[23] S.T. Kent, Protecting externally supplied software in small computers, Ph.D. Dissertation, MIT/LCS/TR-255. MIT, Cambridge, MA, 1980.

[24] R. Kopf, G. Ruhe, A computational study of the weighted independent set problem for general graphs, Foundations of Control Engineering (1987) 167–180.

[25] R.E. Neff, Software piracy: international copyright overview, WESCON/94, Idea/Microelectronics, Conference record, pp. 190–195.

[26] P.M. Pardalos, N. Desai, An algorithm for finding a maximum weighted independent set in an arbitrary graph, Int. J. Comput. Math. 38 (1991) 163–175.

[27] A.D. Rubin, Trusted distribution of software over the internet, Proc. IEEE Symp. on Network and Distributed System Security, pp. 47–53, 1995.

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

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

[30] R.E. Tarjan, A.E. Trojanowski, Finding a maximum independent set, SIAM J. Comput. 6 (3) (1977) 537–546.

[31] B. Venners, The Architecture of Aglets, Java World, ⟨http://www.java-world.com/javaworld/jw-04-1997/jw-04-hood.html⟩, April 1997.

[32] J. Voelker, P. Wallich, How disks are 'padlocked', IEEE Spectrum (1986) 32.

[33] D.J.A. Welsh, M.B. Powell, An upper bound for the chromatic number of a graph and its application to timetabling problems, Comput. J. 10 (1967) 85–86.

[34] S.R. White, L. Comerford, ABYSS: architecture for software protection, IEEE Transactions on Software Engineering 16 (6) (1990) 619–629.

[35] A. Wilson, Software security and the DirectPlay API, Dr. Dobb's Journal (1997) 66.

[36] J. Xue, Edge-maximal triangulated subgraphs and heuristics for maximum clique problem, Networks 24 (1994) 109–120.

[37] X.N. Zhang, Secure code distribution, IEEE Computer 30 (6) (1997) 76–79.