SECURITY POLICY CHECKER AND GENERATOR FOR JAVA MOBILE CODES

Haruhiko Kaiya, Hitoshi Furukawa and Kenji Kaijiri
Faculty of Engineering, Shinshu University, JAPAN
kaiya@cs.shinshu-u.ac.jp cfurukaw@c.cs.shinshu-u.ac.jp kaijiri@cs.shinshu-u.ac.jp

Abstract Java is one of the most famous mobile code systems, and its components can be dynamically downloaded from the other computers over the internet. Because such downloaded components are not always reliable, behaviors of each component are restricted according to the application's policy. However, it is not so easy for the application users or developers to decide the suitable policy. In this paper, we introduce a tool for generating and checking the security policies for Java application. As we deploy Java components spatially on a window of our tool, we can check which component can be executed or not with respect to a set of security policies. In addition, our tool can generate the minimal set of policies to execute all the deployed components.

Keywords: Java Security, Security Policy, Mobile Code

1. Introduction

Most of all computer applications over the internet use software components which can be downloaded from the other machines over the network, and can link the components dynamically during runtime. This kind of components is called *mobile code* [1], which enables applications to change themselves dynamically in runtime.

Mobile codes are useful and easy for application users because the users can use various functions without manual installation, and the codes sometimes contribute to decrease the traffic over the network because the codes are embedded in the client machine and wasteful traffic of data is not occurred in runtime.

The use of mobile codes follows security problems because the use is almost the same that another person directly operates your computers. If an author of a mobile code bears ill will, your data or processing will be damaged by the

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35614-3_21
C. Rolland et al. (eds.), *Engineering Information Systems in the Internet Context*

mobile codes. Therefore, behaviors of mobile codes are normally restricted to a certain extent in some way.

Java is the most famous language with mobile codes, and has simple but powerful security architecture based on the *sandbox model* [2]. In the architecture, no permission is given to all mobile codes by default and permissions granted to specific codes are written as security policies in the client side. However, there is no tool for confirming that such policies meet the security requirements of the client users. In some case, some policy grants over-permissions to a code and the code can sometimes unconsciously destroy the client's resources such as files or processes.

Our tool presented in this paper contributes to confirm the security policies to meet the security requirements of the users intuitively. We assume the following two usages of our tool. First, users of applications with mobile codes may check the security policies against their security requirements before they execute the applications by our tool. If the policies do not meet their requirements, they may request providers of the application or mobile codes to modify the policies or functions of mobile codes. Second, developers of such applications may generate the suitable security policies by our tool. The developers can incrementally modify the policies and select mobile codes used in the applications.

We simply call the application programs that use mobile codes as *clients* in this paper. We also call the user of the clients as *client user*, and the user of our tool as *tool user*. We sometimes use the word *user* if one can clearly distinguish the client user from the tool user.

The rest of this paper is organized as follows. In Section 2, we simply introduce the security architecture of Java2 language system. We briefly introduce related works in Section 3. In Section 4, we show the design and the implementation of our tool. Finally, we summarize our contributions and discuss the future works.

2. Java2 Security System

Java is simple but powerful object-oriented programming language which can use mobile codes. Java can use two kinds of mobile codes, mobile classes and mobile instances. We only focus on the former kind of mobile codes in this paper. Because Java classes are stored or transmitted as *class files* or a *jar archive* which is a composite file of class files and other data, we simply call class files and jar archives as *Java components* or components.

A running instance of a client in a computer is called Java runtime or a runtime in Java system. Java mobile codes can be loaded and linked to a runtime in progress by class loaders, that are part of the runtime. Each class loader enables the runtime to load the components from various sources, e.g file sys-

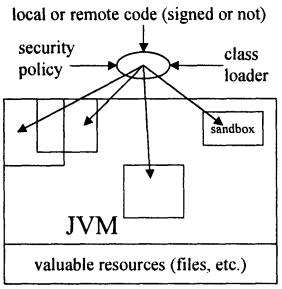

Figure 1. Java2 Security Model[2]

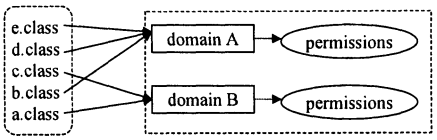

Figure 2. Protection domains[2]

tems, network connections, data bases and teletypewriters of users. Therefore, Java can handle mobile codes from various kinds of sources.

Since JDK1.2, Java has a security model as shown in Figure 1. All Java components are categorized into several kind of sandboxes before the components are executed, with respect to the security policies and class loaders. Class loaders normally mark the attributes on each component. The main attributes are the signature of the component if signed and the location where the component was deployed. The location of codes is normally represented in a URL form such as http://foo.bar/java/.

The valuable resources such as file systems and network connections are guarded by permissions. If a component is executed against the permissions of the component, Java runtime throws exceptions and its execution is not accomplished. Therefore, the role of Java security architecture is to check the permissions granted to each component. Protection domains in Figure 2 are used to this decision.

The components that were deployed in the same location and that are signed in the same signature are categorized into the same group. Such group of components is called a domain or a protection domain in Java system. The mapping from domains to the sets of permissions is written in the security policy, and the policy is normally stored in the file in the client side.

3. Related Work

Bugs in Java security have been reported in many papers [3] since early times. In the content of computer security, the protection from the unauthorized subjects always becomes the central issue[4]. Pure formalization for Java language system and JVM is also contribute to improve the understanding of the security system [5]. Although all these kinds of issues are very important, the suitable usage of the security system is also important issue for Java system. Our tool contributes such suitable usage of Java security system.

Ponder[6] is powerful language for specifying management and security policies. Its framework is large and comprehensive, e.g. ponder can use negative authorization but Java and our tool can only handle the positive one. However, Ponder seems too complex to apply Java application.

4. Design and Implementation of Our Tool

4.1 Requirements

From the overview of Java security architecture in the previous section, we have found that suitable use of security system is not so easy for software users and developers. For example, components in the same domain are granted in the same way because permissions are assigned to each domain. As a result, needless permissions can be given to several components. Permissions for a component are changed when a location or a sign of the component is changed. As a result, a component which was executed in a runtime is not always execute in the same way because of its mobility.

The following functions will contribute to supporting the developers and the users to get the suitable security policies.

1 Developers and users can view the deployment of the Java components over the network at a glance.

2 They can view the functions of each deployed component that can violate the client's security.

3 They can check a policy set whether the set enables the deployed components to be executed or not.

4 They can get a minimal policy set which enables the components to be executed. They may safely adjust a required policy set based on the minimal set.

5 They can analyze the impacts of the changes or mobility of the components.

Based on the requirements above, we have designed and implemented the following tool.

4.2 User Interfaces and Behaviors

Figure 3 shows the overview of our tool. Our tool has four main windows as shown in this figure. The left top window entitled "Virtual Network" shows the deployment of the mobile codes. The client policies are shown in the right top window entitled "Policy File Editor". The left bottom window shows the

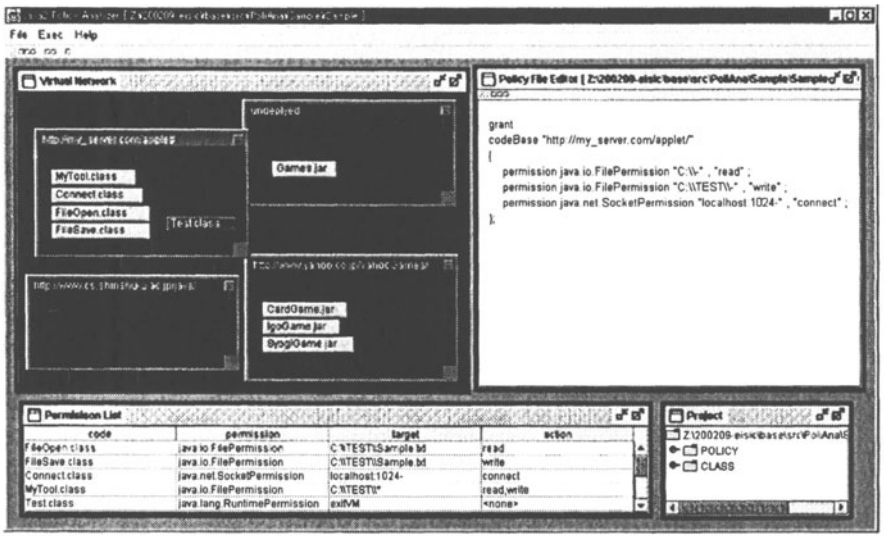

Figure 3. Overview of the Tool

permissions that each Java component needs to be executed, and the right bottom windows entitled "Project" shows a project operated now. In the following parts of this section, we explain each window.

Virtual Network Virtual Network window shows the deployment of mobile codes over the internet. Each sub-window of Virtual Network window represents URL over the internet, and rectangles in each sub-window represent mobile codes deployed in URL corresponding to the sub-window. Though the set of URLs has hierarchical structure, e.g. http:/foo.bar/java/ includes http:/foo.bar/java/com/, we do not represent such structure in Virtual Network window for simplicity.

The color of each rectangle shows the status whether corresponding code can be executed or not. Red color (in monochrome, dark gray color) shows that the code can not be executed, and the gray code (in monochrome, white color) shows that corresponding code can be executed. The URL and the code can be created easily on this tool as shown in Section 4.3.

Permission List Several permissions are necessary for each mobile code to be executed. A window of Permission List shows such permissions of codes. If a code needs more than one permission, more than one line are listed on Permission List for the code. If a code needs no permission, no line is listed.

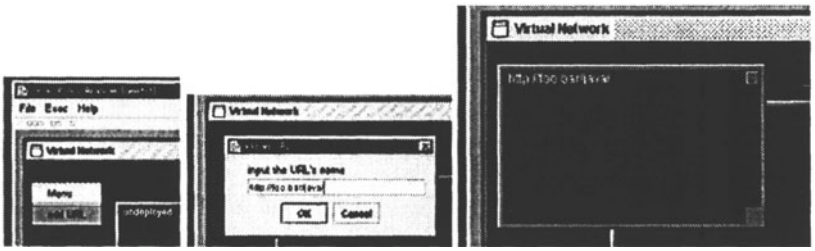

Figure 4. Create a new URL for mobile codes

Currently, permissions of each code should be manually added to the Permission List.

Policy File Editor Users can edit the client's policies on the Policy File Editor. They may write the policies from scratch, or they may edit the minimal set of policies as shown in section 4.4.

A status and requirements of codes are described in both Virtual Network Window and Permission List. Security requirements of client users are described in Policy File Editor. As a result, this tool checks whether descriptions in both Virtual Network Window and Permission List are consistent with descriptions in Policy File Editor or not. In addition, this tool generates consistent descriptions in Policy File Editor with descriptions in both Virtual Network Window and Permission List.

4.3 Basic Operations

Create an URL As shown in Figure 4, users can easily create an URL for mobile codes. First, an user calls pop-up menu on the Virtual Network window and selects "add URL" menu, then the a modal window "add an URL" is appeared. Second, the user fills the name of an URL, then a sub-window corresponding to the URL is appeared.

Deploy a mobile code on an URL As shown in Figure 5, users can deploy a mobile code on a window of an URL. First, an user calls a pop-up menu on a URL window and selects a menu "create code", then a modal window "add a code " is appeared. Second, the user fills the name of a code, then a rectangle corresponding to the code is appeared.

Edit permissions for a mobile code Because the standard permissions, targets and the actions of Java are predefined, such permissions of each

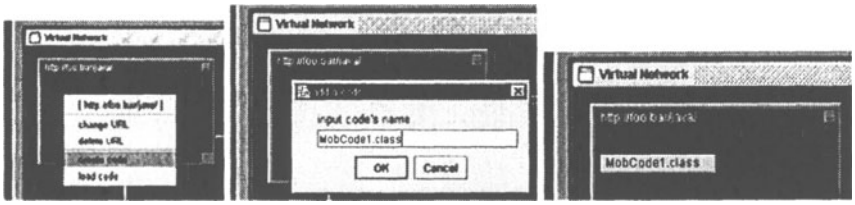

Figure 5. Locate a new code on a location

code can be selected as shown in Figure 6. Tool user may directly write the permissions, targets and the actions while they are not standard.

A typical procedure to add a permission to a code is as follows. First, an user calls a pop-up menu on a rectangle of the code, and selects a menu "add permissions", then a modal window "add a permission" is appeared. On the modal window, the user can select a permission of the code, and he can also select a target and an action of the permission. Several kinds of targets such as path name of a file system can be freely edited by the user.

4.4 Generating a minimal policy set

As shown in Figure 7, our tool can generate a policy set based on the deployment of the codes and the permissions that the codes need. We call such generated policy set as *a minimal policy set*. Currently our tool can not handle the digital signature.

The algorithm to get a minimal policy set is quite simple as follows.

1 For each URL in Virtual Network of our tool, add a string "grant codeBase *URL* {" to the Policy File Editor.

   (a) For each code in the URL,

      i For each permission of the code in Permission List, add a string "*permission action target*" to the Policy File Editor.

2 Add a string "}" to the Policy File Editor.

Because codes deployed in the same location are given the same permissions by a policy set in Java security system, the minimal policy set is not a real minimal set of policies to execute the codes. However, we can preview what kind of permissions are given to the codes in each URL, and can find over-permissions to some codes manually.

For example in Figure 7, although "FilePermission" and "SecurityPermission" are set to "MobCode1.class" and "MobCode3.class" respectively as shown in the Permission List, these two permissions are given to any codes in the location "http://foo.bar/java/" as shown in the Policy File Editor. Therefore, if

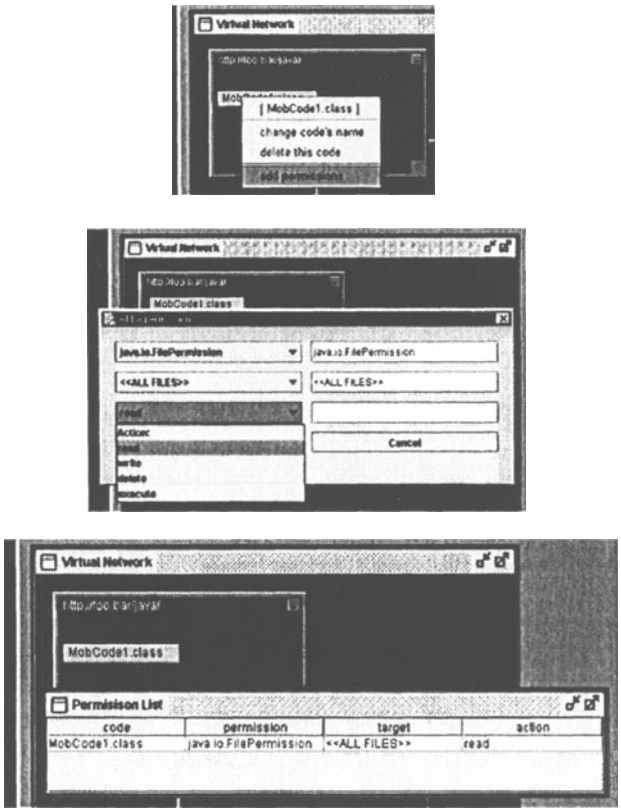

Figure 6. Add a permission to a code

"MobCode3.class" is not enough reliable to have "FilePermission", the client becomes unsecured. Tool user may avoid this situation by moving the "MobCode3.class" to the other or new location.

4.5 Checking the Policy and the Deployments

Our tool can check whether a code can be executed or not against the security policies written in Policy File Editor. Figure 8 shows the example of this kind of check. Although "MobCode1.class" in "http://foo.bar/java/" needs a permission to read all files, Policy File Editor only give the URL a permission to read file "/tmp/". Therefore, "MobCode1.class" can not be executed under this deployment and the policy set.

In Virtual Network, rectangles corresponding to codes which can not be executed are red (in monochrome, it looks like dark gray), and those of executable

Figure 7. Generate a minimal policy set

Figure 8. Check codes' execution against the policy

codes are light gray (in monochrome, white color). We can check whether the policy set is too rigorous for the mobile codes to be executed in the client program.

5. Conclusion

In this paper, we present a graphical tool to check security policies and the deployment of the codes. The tool also generate a minimal policy set which gives the necessary permissions to the mobile codes.

Currently, our tool do not support jar archives and the digital signature. Three dimensional representation of Virtual Network seems to be suitable to handle the digital signature.

Hierarchical structure of the URL and the target such as file names is not also supported by our tool. Such kind of structure will partially contribute for improving the scalability of our tool. However, we should carefully introduce such structure into our tool. For example, suppose the permissions of "http://foo.bar/java/" are subset of the permissions of "http://foo.bar/java/afo/". A hierarchical structure between such URL's is no sense because no more permissions are granted to the codes deployed in "http://foo.bar/java/afo/" by this structure.

Permissions needed by each component are manually picked up by tool users now. However, these can be automatically picked up from source codes or class files, because security related codes transitively use a code which includes the Permission class or its sub-classes. As Java class files can be easily disassembled, it is not obstacle that the source codes are not published.

Most serious problem of a current version of our tool is that our tool can not directly refer to the security requirements of client users. Because descriptions in a policy file only tell what are granded, no one can directly know such policies contribute for satisfying the requirements of client users. In our another work[7], we present a way to formally specify the security requirements and to verify the security requirements written in Z notation. We want to join this work together with our tool.

References

[1] Tommy Thorn. Programming languages for mobile code. *ACM Computing Surveys*, 29(3):213–239, Sep. 1997.

[2] Sun Microsystems, Inc. *Java Security Architecture (JDK1.2)*, Oct. 1998. Version 1.0.

[3] Drew Dean, Edward W. Felten, and Dan S. Wallach. Java Security: From HotJava to Netscape and Beyond. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 190–200, May 1996.

[4] Vincent Tam and Rakesh K. Gupta. Using Class Decompilers to Facilitate the security of Java Application. In *WISE'00 proceedings*. IEEE, 2000.

[5] T. Jensen, D. Le Metayer, and T. Thorn. Security and Dynamic Class Loading in Java: A Formalization. In *Proceedings of International Conference on Computer Languages*, pages 4–15, May 1998.

[6] Ponder homepage. `http://www-dse.doc.ic.ac.uk/Research/policies/ponder.html`.

[7] Haruhiko Kaiya and Kenji Kaijiri. Specifying Runtime Environments and Functionalities of Downloadable Components under the Sandbox Model. In *International Symposium on Principles of Software Evolution*, pages 138–142, Kanazawa, Japan, Nov. 2000. IEEE Computer Society Press.