A SUPPORTING TOOL TO IDENTIFY BOTH SATISFIED REQUIREMENTS AND TOLERANT THREATS FOR A JAVA MOBILE CODE APPLICATION

Haruhiko Kaiya, Kouta Sasaki, Chikanobu Ogawa and Kenji Kaijiri
Shinshu University
4-17-1 Wakasato, Nagano, 380-8553 JAPAN

Keywords: Security Policy, Security Requirements, Java, Requirements Analysis.

Abstract: A mobile code application can be easily integrated by using existing software components, thus it is one of the promising ways to develop software efficiently. However, using a mobile code application sometimes follows harmful effects on valuable resources of users because malicious codes in such an application can be activated. Therefore, users of mobile code applications have to identify both benefits and risks by the applications and to decide which benefits should be gotten and which risks should be tolerated. In this paper, we introduce a tool to support such users. By using this tool, the users can identify security related functions embedded in each mobile code automatically. The users can also relate these functions to each benefit or risk. By defining a security policy for mobile codes, some functions are disabled, thus some benefits and risks are also disabled. By adjusting the security policy, the users can make decision about the benefits and the risks.

1 INTRODUCTION

Mobile code technology is useful because it is easy to integrate a software service on the fly. It is also easy to maintain and update mobile code components in such a service because codes are basically downloaded and linked in its runtime. In addition, alternative codes can be easily selected for meeting requirements changes because we can reuse fine-grained software components in ad hoc manner. For example, suppose there are many alternative codes for data communication, and their efficiency and license cost are different with each other. An integrator will select a code that is not so fast but cheap normally, but he/she in urgent situation can replace the code into another that is very fast but expensive on the fly.

However, there are several problems in using mobile codes, and one of the significant problems is about malicious codes. If behaviors of malicious codes are not restricted, valuable resources can be leaked and/or destroyed. For example, your credit card information could be stolen. We call such harmful effects by malicious codes as threats in this paper. Therefore, we have to identify which requirements should be satisfied and which threats should be avoided when we integrate a mobile code application. In addition, we think it is impossible both to satisfy all requirements and to avoid all threats completely. In fact, we have compromised with software systems with unsatisfied requirements and tolerant threats. Therefore, we have to also decide trade-offs between satisfied requirements and tolerant threats.

We have already proposed a method to identify trade-offs between them caused by Java mobile code applications (Kaiya et al., 2003; Kaiya et al., 2004). However, we cannot effectively examine our method without supporting tools, because the method requires tiresome but systematic tasks. In this paper, we will introduce a supporting tool and the plan to apply the tool into security education.

The rest of this paper is as follows. In the next section, we briefly introduce the mechanism of Java security architecture. In section 3, we will explain how to use our tool. In section 4, we will explain why, where and how to apply our tool. Finally, we conclude our current results and summarize future plan.

2 JAVA SECURITY

Java security architecture is based on the sandbox security model (Java, 1998). There are many security related features in Java security architecture, but we only focus on the permission and the security policy.

Kaiya H., Sasaki K., Ogawa C. and Kaijiri K. (2006).
A SUPPORTING TOOL TO IDENTIFY BOTH SATISFIED REQUIREMENTS AND TOLERANT THREATS FOR A JAVA MOBILE CODE APPLICATION.
In *Proceedings of the Eighth International Conference on Enterprise Information Systems - ISAS*, pages 444-448
DOI: 10.5220/0002445604440448

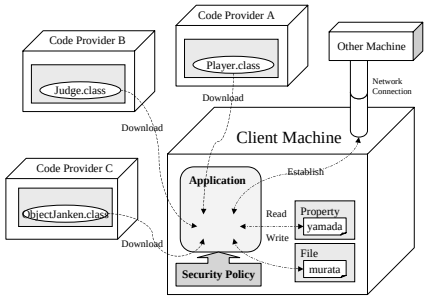

Figure 1: Environment for a Java application.

Each permission correspond to the right to access system resource(s) such as files, network connections, running processes and so on. To grant the pieces of the right to a Java application, the security policy is given to the application. Figure 1 shows an example of the environment for a Java application. An application in this figure consists of three pieces of codes and it accesses a file and a property, and establishes a network connection to another machine.

When inadequate policy is given to an application, malicious codes can be activated, thus security threats can be made. For example, data can be illegally leaked because the policy inadequately grants the right for files and network connections.

To avoid the threats caused by malicious codes, codes are distinguished with respect to both the site where a code is placed and the signature, and restricted in different ways. If a code is downloaded from a trusted site or a trusted agent signed the code, we may believe the code does not cause any threats.

However, we cannot or do not always use only trusted codes in fact, e.g., some kinds of free software. In addition, even the trusted codes cause security threats because of their bugs or our inadequate usage. Therefore, application integrators have to investigate which requirements are satisfied and what kinds of threats can be caused by a mobile code application by themselves.

3 USAGE OF OUR TOOL

This tool supports application integrators to identify which requirements are satisfied by a mobile code application. In addition, the tool also supports to identify threats caused by the application. Because reuse of mobile codes is intended, threats can be avoided by tightening up the security policy and/or by replacing a mobile code including malicious parts with another compatible code. In some cases, some requirements cannot be satisfied because of the tightened policy, thus we have to sometimes give up some requirements or to accept the threats. This tool also supports to find such trade-offs.

3.1 Major Functions

Our tool provides the following six functions. By using such functions in a requirements analysis process, integrators can identify the achievement of requirements and threats, and decide trade-offs between requirements and threats.

3.1.1 Network Deployment Function

Our tool consists of several internal windows as shown in Figure 2, and a top left window called “Virtual Network Window” is an analogical model of a deployment of computers each of which provides mobile codes. In addition, permissions that are required by such codes are automatically extracted from source codes or byte codes, and listed in a middle window called “Permission Table”. XML based representation for Java source codes (JavaML) and its supporting tool [1] and Java decompiler [2] are used in this function. By using this function, users can understand what kinds of security related functions could be activated by each mobile code.

3.1.2 Policy Edit Function

Based on the deployment of mobile codes shown in the “Virtual Network Window”, our tool can automatically generate a security policy that grants all permissions required by all codes. The generated policy is shown in the right top window in Figure 2, and users can freely edit the policy. Users may also edit a policy from scratch, but users can easily arrive at intended policy by removing the granted permissions from the generated policy.

3.1.3 Policy Check Function

According to the policy in “Policy Editor”, each permission in “Permission Table” is automatically checked whether it can work or not under the policy. The column labeled by “Check” in “Permission Table” shows the results.

3.1.4 Requirements Edit Function

Users can list their requirements on the left bottom window called “Requirements Window” in Figure 2. The requirements are simply itemed as shown in the figure. Each requirement is basically categorized into the following four types.

[1] http://www.badros.com/greg/JavaML/
[2] http://jode.sourceforge.net/

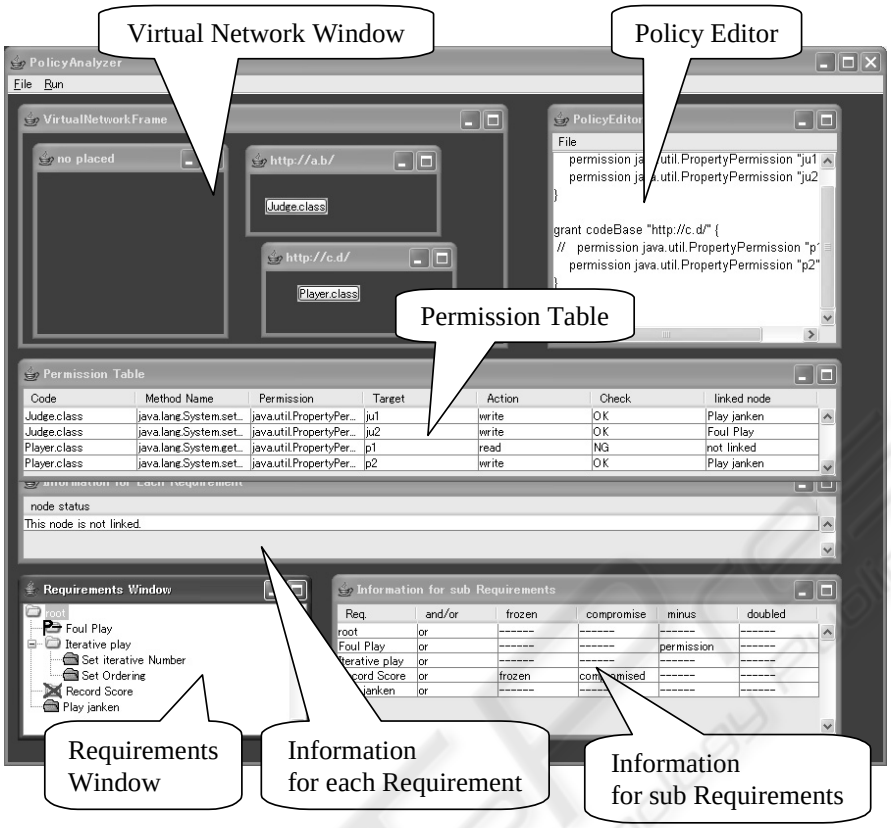

Figure 2: A Snapshot of Our Tool.

1. Unrelated: A requirement which is not related to security related permissions.
2. Granted: A requirement which requires security related permissions, and the permissions are granted in a current situation, i.e., code deployment and policy.
3. Revoked: A requirement which requires security related permissions, but the permissions is not granted. As a result, the requirement is given up.
4. Threatening: When there are some granted permissions that are not required by requirements and such permissions could enable some mobile codes to perform unintended and harmful functions, we call such functions as threats or threatening requirements. Threats are also listed in the "Requirements Window".

The graphical icon for each requirement on "Requirements Window" is decided according to the type of the requirement.

3.1.5 Requirements Check Function

Users can decide the types of each requirement. If users identify that a requirement requires permission(s) on the "Permission Table", users can make relationship between the requirement and the permission(s). Related requirements for each permission can be found in the right most column on the "Permission Table". Related permissions for each requirement can be found in a window called "Information for each Requirement".

A requirement is regarded to be satisfied when its related permissions are all granted. On the other hand, a requirement is regarded to be revoked one when there is a permission that is required by the requirement but is not granted in the current situation.

3.1.6 Threat Check Function

Some granted permissions sometimes do not relate to any requirements. Users can explore the possibility of threats by examining such permissions and their effects. If a threat can be activated by such permissions, users can add a threat on the "Requirements Window", and make relationship between the requirement and the permissions. The threat is regarded to be satisfied when current permissions enable the threat to be made.

3.2 Typical Process

By removing some lines from a policy e.g., in "Policy Editor" in Figure 2, granted permissions are decreased in general, and vice versa. When granted permissions are changed, satisfied requirements and threats are also changed. The requirements, threats and the policy are decided by repeating such changes. Finally, integrators have to decide which requirements may be abandoned and which threats can be tolerable with respect to the users' needs.

4 APPLICATION IN EDUCATION

Even though we frequently meet real threats for computers thanks to the Internet, there are few educational exercises for security threats and their avoidance at least in our country. Consequently, students do not clearly understand the importance of security requirements. On the other hand, our requirements for software can be rarely satisfied without security related functionalities, thus we have to find trade-offs between our requirements and threats that should be accepted.

By using Java, we can easily implement and activate malicious mobile codes, and also avoid such threats by using Java security policy mechanism. In fact, a bootstrap program for Java mobile code application can be written in less than 150 lines of code. Thus, Java mobile codes and security system seems to be suitable for students to learn the importance of security requirements.

We have planed to put a course into practice for students. The objective of the course is to make our students to understand the importance for analyzing security requirements and threats. Another objective is to confirm the usefulness of our tool, e.g., whether our tool contributes to identify requirements and threats and to find tradeoffs among them.

5 CONCLUSIONS

In this paper, we introduced a supporting tool to identify both the achievement of requirements and threats for Java mobile code application. We have applied our tool in security requirements education.

Mobile codes seem to be a little bit singular, but we can find important application areas for mobile codes. For example, embedded systems normally have only limited resources, such as disks, thus, mobile codes can be applied to such domain. Currently, we mainly intended to use our tool in educational settings. However, we believe our tool can be also applied to practical software integration.

There are several limitations of our tool. One limitation comes from the intended platform, Java. Java security policy is too simple to specify complex security policy, e.g., we cannot write revoke rules in a policy. Because of such simplicity, we cannot apply our tool to generic software systems. Another limitation is its usability. Users of our tool do not have to know Java language itself, but they have to know the syntax of security policy, even though the policy syntax is much easier than Java's. Finally, our tool does not explicitly support the selection of alternative mobile codes now. This is a problem when we apply our tool into practice, thus we will extend our tool from this point. However, we do not think it is a problem in our educational setting mentioned in 4 because one of the objectives of our educational course is to make our students to understand the importance of analyzing security requirements and threats.

Threatening requirements mentioned in 3.1.4 are very similar to obstacles in KAOS (van Lamsweerde, 2004). Their difference is that threatening requirements do not have to obstruct existing requirements but obstacles are basically identified by obstructing existing requirements or goals. Thus, threatening requirements in this paper is not easy to be identified by KAOS approach. Misuse case approach is also useful method to identify security requirements, but its weakness was argued in (Sindre and Opdahl, 2005). Our tool can partially overcome such weakness, for example, the process navigated by our tool is not open-ended but systematically terminated if the user can compromise on a specific policy and its consequences; giving up requirements and/or accepting threats. Software Fault Tree (Helmer et al., 2002) is also systematic approach, but it is specialized for the requirements analysis of intrusion detection systems. A system called SoftwarePot (Kato and Oyama, 2003) can be also applied to the problems we focused. In SoftwarePot, applications are executed in some kind of sandbox, and users have to decide whether an access to the valuable resources should be granted or not in each time. We think SoftwarePot approach seems to be practical, but it does not contribute to improving users' understanding about security problems.

REFERENCES

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

Helmer, G., Wong, J., Slagell, M., Honavar, V., Miller, L., and Lutz, R. (2002). A Software Fault Tree Approach to Requirements Analysis of an Intrusion Detection System. *Requirements Engineering*, 7(4):207 – 220.

Kaiya, H., Sasaki, K., and Kaijiri, K. (2004). A Method

to Develop Feasible Requirements for Java Mobile Code Application. *IEICE Trans. Inf. and Syst.*, E87-D(4):811–821.

Kaiya, H., Sasaki, K., Maebashi, Y., and Kaijiri, K. (2003). Trade-off Analysis between Security Policies for Java Mobile Codes and Requirements for Java Application. In *11th IEEE International Requirements Engineering Conference*, pages 357–358.

Kato, K. and Oyama, Y. (2003). SoftwarePot: An Encapsulated Transferable File System for Secure Software Circulation. *Lecture Notes in Computer Science*, 2609:112 – 132.

Sindre, G. and Opdahl, A. L. (2005). Eliciting security requirements with misuse cases. *Requirements Engineering*, 10(1):34 – 44.

van Lamsweerde, A. (2004). Elaborating Security Requirements by Construction of Intentional Anti-Models. In *Proceedings of ICSE'04*, pages 148–157.