# PORTAM: Policy, Requirements and Threats Analyzer for Mobile Code Application

Haruhiko Kaiya Kouta Sasaki Kenji Kaijiri
Dept. of Computer Science, Shinshu University
4-17-1, Wakasato, Nagano 380-8553, Japan
http://www.cs.shinshu-u.ac.jp/~kaiya/

## Abstract

*Users and providers of an information system should clearly understand the threats caused by the system as well as clarify the requirements for the system before they use the system. Especially, they should be very careful when they use a system with components and/or services provided by third parties. However, there are few methods or tools to learn and confirm such issues. In this paper, we present a supporting tool called "PORTAM" for such users and providers to understand the threats and the requirements. Suppose some requirements cannot be satisfied when some threats are avoided, and vice versa. In such cases, they should decide whether the requirements should be satisfied or the threats should be avoided. The tool also helps them to decide such kinds of trade-offs. Current version of this tool handles Java mobile code applications, thus users of our tool can easily feel real threats. Although the current version deals only with Java components, the ideas behind the tool can be applied to software in general. We finally report experimental results to confirm the usefulness and the educational effects of this tool.*

## 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. Unfortunately, we do not explicitly understand the importance of specifying requirements and threats of an information system even through we meet actual threats every day via Internet.

We have already proposed a method to identify trade-offs between them caused by Java mobile code applications [12, 11]. 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 results to apply the tool into security education. Main contribution of this paper is to show how far security requirements analysis can be supported and how far security requirements education can be accelerated by this kind of tool.

Security issues in requirements engineering are widely focused recently [6] because of the wide use of Internet, and there are many researches of this issue. One of the characteristics of our tool is its simplicity, and the tool easily enables learners and/or students to learn the importance of security requirements in the classroom. For example, learners can meet real threats caused by an application vividly and they can analyze the reasons by using our tool shortly. This kind of activities will largely improve their understanding of security requirements. This hypothesis is partially confirmed in our case study. There are many complex and complete models/tools for security requirements but it is not so easy for learners to have vivid experiences. This is one of the reasons why our tool is limited to handle Java mobile code applications.

The rest of this paper is as follows. In the next section, we explain the mechanism of Java mobile code applications. Although Java system is too simple, typical security problems can be handled within Java system. Section 3 summarize the requirements of our analysis tool based on its previous discussion. In section 4, we introduce our tool in detail. In section 5, we report a case study to confirm the usefulness and educational effects of our tool. In section 6, related works are discussed, and finally we conclude our

Proceedings of the Sixth International Conference on Quality Software (QSIC'06)
0-7695-2718-3/06 $20.00 © 2006 IEEE

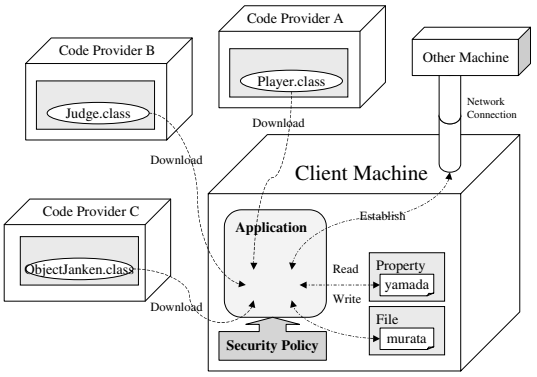


**Figure 1. Environment for a Java application**

current results and show future works.

## 2. Mobile Code Application

In this section, we explain security architecture of Java system. Although Java security architecture is very simple, it supports fundamental security issues such as integrity, confidentiality and availability. Therefore, Java system is suitable enough to learn the importance of security requirements.

Java security architecture is based on the sandbox security model [17]. There are many security related features in Java security architecture, but we only focus on the permission and the security policy. 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 other machine.

When inadequate policy is given to an application, malicious codes can be activated, thus security threats can be made. The examples of threats for each type of security issues are as follows.

- Integrity: Files or system properties are illegally modified because the security policy inadequately grants the right for files or system properties.
- Confidentiality: Data are illegally leaked because the policy inadequately grants the right for files and network connections.
- Availability: Calculation is illegally aborted because the policy permits the right for killing the calculation process.

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 and users have to investigate which requirements are satisfied and what kinds of threats can be caused by a mobile code application by themselves.

We assume a supporting tool is needed and/or useful to investigate such things. The usage of our tool is explained in detail in section 4.3, and its evaluation is reported in section 5. We also assume archiving such investigation improves their understanding about mobile codes and security. A part of a case study in section 5 was designed for confirming this assumption.

## 3. Requirements for A Supporting Tool

Based on our previous works [12, 11] and the discussion in the previous sections, we specify the requirements for the tool as follows.

1. The tool shall support requirements analysis for a Java mobile code application.
2. The tool shall manage the requirements for an application and the threats by the application. A threat is an inconvenient result caused by the mobile code application.
3. The tool shall manage permissions that are required to satisfy each requirement.
4. The tool shall manage permissions that enable a threat to be activated.
5. The tool shall manage available mobile codes.
6. The tool shall be able to extract security related permissions from each mobile code.
7. The tool shall be able to generate security policy that grants permissions in available mobile codes.
8. The tool shall be able to modify security policy to satisfy requirements and to avoid threats.
9. The tool shall be able to check which requirements are satisfied or not, and which threats are avoided or not under a security policy.
10. The tool shall enable its user to abandon some requirements and/or to accept some threats.

## 4. Overview of PORTAM

This tool supports application integrators and/or users 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.

### 4.1 Major Functions

Our tool mainly provides the following six functions. By using such functions during a requirements analysis process, integrators and/or users can identify the achievement

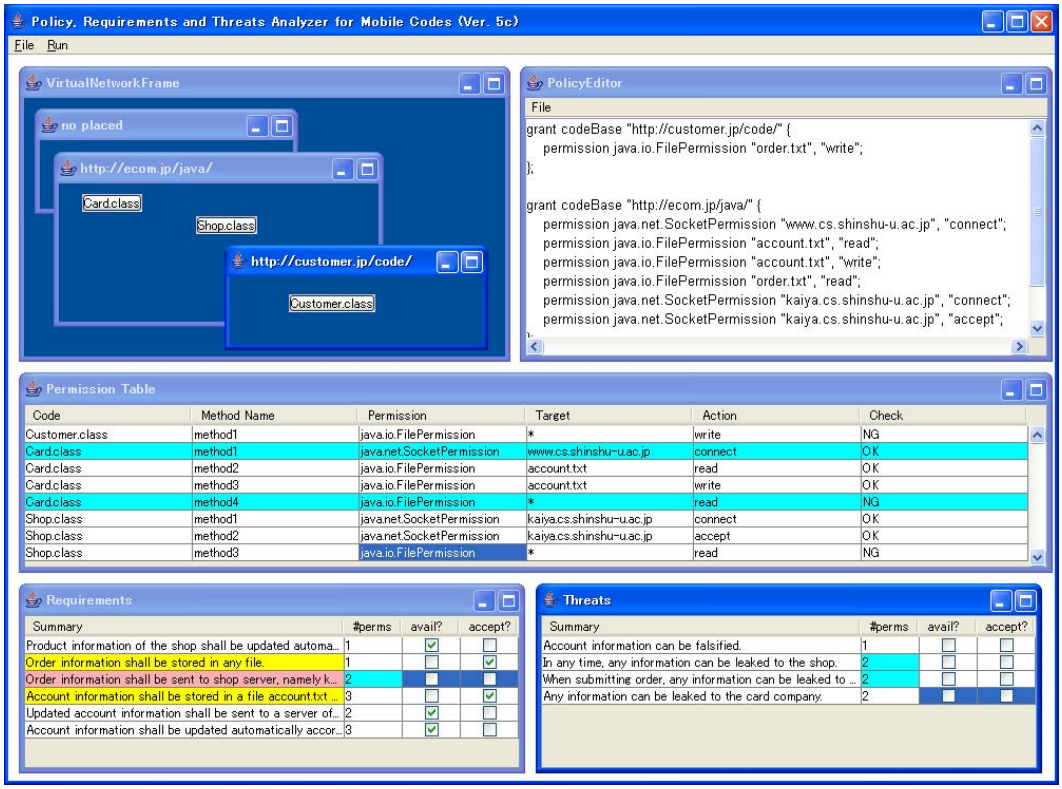


**Figure 2. A Snapshot of PORTAM**

of requirements and threats, and decide trade-offs between requirements and threats.

#### 4.1.1 Network Deployment Function

Our tool consists of several internal windows as shown in Figure 2, and a top left window called "Virtual Network Frame" 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 semi-automatically extracted from source codes or byte codes, and listed in a middle window called "Permission Table". By using this function, users can understand what kinds of security related functions could be activated by each mobile code.

Because permissions are extracted by using a static source code analysis, extracted permissions are not always used in each runtime. In addition, targets in each permission such as file names or host names sometimes cannot be extracted automatically because targets are sometimes represented as variables in source codes. The tool user has to edit targets manually in such cases.

#### 4.1.2 Policy Edit Function

Based on the deployment of mobile codes shown in the "Virtual Network Frame", our tool can automatically generate a security policy that grants all permissions required by all codes. The generated policy is put in the right top window called "Policy Editor" in Figure 2, and users can freely edit the policy. The policy shown in Figure 2 is already edited so as to revoke some permissions. 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.

#### 4.1.3 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. For example in Figure 2, first, fifth and the last permissions are marked as "NG" thus they cannot work under the policy in Figure 2.

In addition, we can easily identify which requirements and/or threats use permission. By clicking a row in "Permission Table", requirements and threats in "Requirements" and "Threats" windows are colored. For example in Figure 2, the tool tells us that the last permission in "Permission Table" is used in the third requirement in "Requirements" window, second and third threats in "Threats" window when we click a row in "Permission Table" corresponding to the permission. The "#perms" column of rows in "Requirements" and "Threats" windows corresponding to the requirement and the threats is colored in blue.

#### 4.1.4 Requirements and Threats Edit Function

Users can list their requirements on the left bottom window labeled "Requirements" in Figure 2. The requirements are simply itemed as shown in the figure. Users can also list their identified threats on the right bottom window labeled "Threats" in the same way. Each requirement or threat has the following properties.

- A list of permissions that are required by the requirement or the threat. A requirement is satisfied or a threat can be activated when such corresponding permissions all work. A tool user should specify the list of permissions manually because it is hard to evaluate the meaning of such list automatically.

  As mentioned in 4.1.1, permissions are not always used in each runtime because of the static analysis. Therefore, the list of permissions does not always guarantee that a requirement is satisfied or a threat is activated. However, the list of permissions helps a requirements analyst to explore the possibility of threats because such permissions are sometimes used in runtime.
- A truth value whether a requirement or a threat can be activated. If all permissions above are "OK", the value is true, otherwise false.
- A truth value whether the user abandons the requirement or accepts the threat.

These properties are shown on the GUI of PORTAM. For example in Figure 2, a window "Threats" is focused and four threats are listed. The last threat "any information can be leaked to the card company" is focused and the threat is related two permissions. The permissions are the second and fifth ones, and they are colored in a middle window "Permission Table". Because the fifth permission in "Permission Table" is "NG", the threat cannot be activated. Thus the check box "avail?" of the threat is not checked. We do not have to accept this threat because the threat cannot be activated in this situation.

#### 4.1.5 Requirements Check Function

Whether a requirement is satisfied or not is decided according to the status of permissions related to the requirement. There are three status of a requirement and the status is shown by color on GUI.

- Satisfied status (white): The corresponding requirement will be satisfied because all related permissions can be permitted.
- Accepted status (yellow): Although the requirement cannot be satisfied by revoked permissions, the user decides to abandon the requirement.
- Unstable status (pink): The requirement cannot be satisfied now but the user does not accept the fact. The user has to decide to accept the fact or to modify security policy to change the status of the requirement.

To finish the requirements analysis, requirements in unstable status should be eliminated completely. In Figure 2, there are six requirements. Second and forth requirements are colored in yellow thus they are accepted. Third requirement are colored in pink thus it is unstable.

#### 4.1.6 Threat Check Function

Whether a threat can be avoided or not is also decided according to the status of permissions related to the threat. There are also three status of a threat and the status is shown by color on GUI.

- Avoided status (white): The corresponding threat will be avoided because at least one of related permissions cannot be permitted.
- Accepted status (yellow): Although the threat can be activated by all granted permissions, the user decides to accept the threat.
- Unstable status (pink): The threat can be activated now but the user does not accept the fact. The user has to decide to accept the fact or to modify security policy to change the status of the threat.

In the same way as the requirements check, threats in unstable status should be eliminated completely to finish the requirements analysis. In Figure 2, there are four threats, and they are all in avoided status.

### 4.2 Technologies used in our Tool

To extract permissions required by a code, we have to analyze the code, especially method calls from the code to other codes. Security related permissions are required only when specific methods in Java standard API are called. Thus, it is sufficient for our tool to identify method calls from the code to such API methods. Our tool achieves a static source code analysis by using XML based representation for Java source codes (JavaML). Our tool invokes an extended version of a Java compiler called jikes [1] to convert a Java source code to a JavaML representation. The converter is not written in Java, thus our tool invokes it as an external program.

Our tool can handle Java class files (byte codes) by using decompiler called jode [2]. Our tool is developed as a Java application and decompiler jode is also written in Java, thus it is easy to be invoked from our tool.

### 4.3 Typical Processes

One of the typical processes by using this tool is to explore threats and policy under given mobile codes and requirements. A case study in the next section is categorized in this type. According to the codes and their deployment, our tool can list the permissions in the codes on "Permission Table". By using requirements and threats edit function mentioned in 4.1.4 of our tool, each requirement is related to several permissions. A user should explore threats to browse the list of permissions, and each threat is also related to several permissions. By using policy edit function mentioned in 4.1.2 of our tool, a security policy that grants all permissions can be generated on "Policy Editor". By removing some lines from a policy in "Policy Editor", granted permissions are decreased in general, and vice versa. When

[1] http://www.badros.com/greg/JavaML/

[2] http://jode.sourceforge.net/

granted permissions are changed, satisfied requirements and threats are also changed. The requirements, threats and the policy are decided by repeating such changes. Finally, a user has to decide which requirements may be abandoned and which threats should be accepted with respect to needs of the application users.

Another typical process is to explore suitable mobile codes and policy under several requirements and threats. To satisfy a requirement, several permissions are required. A user drops existing mobile codes on "Virtual Network Frame" to explore required permissions. To avoid a threat, some permission should not be activated. A user writes security policy not to activate such permissions.

## 5. Case Study

The objective of this case study is to confirm the usefulness of our tool. If the following metrics are improved by using our tool, we may assume our tool is useful in requirements elicitation.

- The amount of threats to be found.
- The amount of fatal threats to be found. Fatal threats are defined for each exercise and subjects off course do not know them before performing the exercise.
- The amount of wrong threats to be found. We decide a threat is wrong when the threat cannot be activated under given permissions. Because we cannot objectively decide that a threat is really inconvenient for the subject, we do not take such inconvenience into account.
- The efficiency of an elicitation task.

As explained in 4.1, our tool cannot find threats automatically but only support an analyst to find threats. Therefore, this kind of an experimental study is required. Based on the assumptions above, we design an experiment as follows.

Another objective of this case study is to confirm the educational effects by analyzing requirements and threats. As mentioned in the introduction, we do not clearly understand the importance of specifying both requirements and threats of an information system even though we meet actual threats everyday via Internet. After performing the experiment, we sent out questionnaires to confirm the educational effects. The contents, results and discussion are shown in 5.4.

### 5.1 Design of an Experiment

We perform a comparative experiment by using several numbers of subjects. Each subject performs the following six exercises in turn.

**S1** Learn mechanism of security policy to execute a mobile code application. Because most subjects are unfamiliar with mobile code applications, they have to become familiar with such applications through this exercise.

**S2** Write their own security policy to satisfy given requirements. In this exercise, the existence of threats is secret before the answer is shown. From this experiment, subjects can understand the existence of potential threats in mobile codes.

**S3** Write their own security policy to satisfy given requirements and to avoid given threats.

**S4** Perform the same exercise by using our tool. This exercise is simply used for subjects to learn how to use the tool.

**S5** Write a security policy to satisfy given requirements and find threats. Note that the codes, their deployment and the policy that grants all permissions in the codes are also given. Half of subjects are permitted to use our tool but another half are not. Whether a subject is permitted to use a tool or not is decided at random.

**S6** Perform the same kind of an exercise as S5 by using other requirements and codes. Subjects using our tool in S5 are not permitted to use our tool but the others are.

The data only in exercises S5 and S6 are used in our analysis because other exercises are basically used for subjects to be familiar with mobile code applications and our tool. For each result of exercises S5 and S6 of each subject, we count the following values.

- The number of threats that are found by the subject.
- The number of fatal threats.
- The number of wrong threats.
- The spending minutes to solve the exercise.

By comparing the results of subjects using our tool to the results of subjects without tool, we confirm the following assumptions.

1. Subjects with our tool can find more threats than subjects without the tool.
2. Subjects with our tool can find more fatal threats than the others.
3. Subjects with our tool can find fatal threats frequently than the others. In other words, the ratio of fatal threats found by the subjects is higher than the ratio by the others.
4. Subjects without our tool write more wrong threats than the others.
5. Subject without our tool write wrong threats frequently than others. In other words, the ratio of wrong threats by the subjects without our tool is higher than the ratio by the others.
6. Subjects with our tool can perform the exercises more efficiently than the others.

### 5.2 Results

We achieved this experiment in a course of our university. The course took five weeks, thus exercises S3 and S4 were performed in the same week. In each week, we spent three hours on classwork including exercises. To improve the understanding of mobile codes applications, attendance check and answer submission were achieved by mobile code applications. About 30 third grade bachelor students were participated in this course. They have already studied software engineering fundamentals and Java programming. We selected 20 students as our subjects based on the following conditions.

- The student performed all six exercises.
- The student agreed on the fact that his/her data were used in our research.

In exercise S5, each subject analyzed a shopping system via Internet. Five requirements were shown and this exer-

**Table 1. Results of S5 (Average per subjects)**

| | tool | manual |
|---|---|---|
| Number of threats (X) | 2.9 | 3.3 |
| Number of fatal threats (Y) | 1.4 | 1.0 |
| Number of wrong threats (Z) | 0.6 | 0.7 |
| Ratio of fatal threats (Y/X) | 0.48 | 0.30 |
| Ratio of wrong threats (Z/X) | 0.206 | 0.212 |
| Spending minutes | 167 | 168 |

**Table 2. Results of S6 (Average per subjects)**

| | tool | manual |
|---|---|---|
| Number of threats (X) | 2.5 | 3.5 |
| Number of fatal threats (Y) | 0.7 | 0.8 |
| Number of wrong threats (Z) | 0.4 | 1.3 |
| Ratio of fatal threats (Y/X) | 0.28 | 0.22 |
| Ratio of wrong threats (Z/X) | 0.16 | 0.37 |
| Spending minutes | 161 | 153 |

cise had the following three fatal threats.

1. An account can be falsified.
2. User information can be leaked to shops.
3. User information can be leaked to credit card company.

In exercise S6, each subject analyzed an e-learning system via Internet. Six requirements were shown and this exercise had the following two fatal threats.

1. The correct answer can be leaked to the other learners.
2. A learner can unfairly read the others answer (cheating).

Tables 1 and 2 show the results. In each experiment, half of subjects used our tools but others did not. If a subject used our tool in S5, the subject did not use the tool in S6 and vice versa. The column labeled by "tool" in the tables is the data of subjects using our tool. On the other hand, the column labeled by "manual" is the data of subjects without our tool. Each result is the average of each type of subjects. For example in experiment S5, a subject found 1.4 fatal threats in average with our tool. Because the number of fatal threats in S5 is three, a subject found only half fatal threats in average.

### 5.3 Discussion

As shown in Tables 1 and 2, not all assumptions in 5.1 are confirmed. Especially, subjects without tool ("manual" column) found more threats than the others. However, subjects without tool tended to write more wrong threats than the others as shown in third and fifth rows in Tables 1 and 2. Thus our tool seems to contribute to the accuracy for finding threats. Although the ratio of wrong threats is relative low (0.2 and 0.16), we investigate the reason of the wrong threats. When a subject with our tool finds and specifies a threat, the subject has to relate the threat with several permissions. We assume there is a gap between the threat and the permissions because permissions show the concepts at the implementation level but the threat does not. Our tool has to support stopping such gap.

The tool also seems to contribute to finding fatal threats a little bit based on the fourth row "the ratio of fatal threats" in the tables. By investigating the contents of the threats written by subjects, a function making relationships between a threat and permissions seems to contribute to find fatal threats because the combination of permissions causes fatal threats. We will discuss how to improve our tool based on discussion here in the final section.

### 5.4 Questionnaires

Our questionnaires consist of the following three kinds of issues. Each issue has several questionnaires as follows.

- Issues in a mobile code application itself.
  - M1. Can you understand the running mechanism of mobile code applications? [yes/no]
  - M2. Can you understand the role of security policy? [yes/no]
  - M3. Do you wish to use mobile code applications in the future? [yes/no]
  - M4. Do you wish to develop mobile code applications in the future? [yes/no]
- Issues in requirements analysis for mobile code applications.
  - R1. Do you begin to take care when you use libraries or programs provided by the others? [yes/ever since/no]
  - R2. Can you understand the existence of threats in mobile code applications? [yes/no]
  - R3. Can you understand that there are sometimes trade-offs between satisfying requirements and avoiding threats? [yes/no]
  - R4. Do you think compromises are sometimes necessary in requirements and threats analysis? [yes/no]
  - R5. Do you begin to think users of an information system should be identify both requirements for the system and threats by the system? [yes/ever since/no]
  - R6. Do you begin to think developers of an information system should be identify both requirements for the system and threats by the system? [yes/ever since/no]
- Issues about our supporting tool.
  - T1. Can you understand how to use our tool? [yes/no]
  - T2. Did our tool help you to find threats and/or forgotten requirements? [yes/partially/no]

Note that "ever since" means that the subject thought or understood the issue before this experiment.

Table 3 shows the results of questionnaires. Our subjects understood the mobile code application itself, but some of them worried about unidentified threats. Another complained that he did not always connect to Internet and he did not always use mobile code applications. There is a kind of mobile codes application that can work under disconnected situation, but we did not mention such kind of application in this course. We have to introduce such application by eliminating such complaints. Most subjects mentioned easiness to update software or usefulness as positive reasons with respect to users. With respect to developers, more subjects gave negative answers. Typical reasons were as follows; it

**Table 3. Results of Questionnaires**

| | yes | ever since | no |
|---|---|---|---|
| M1 | 17 | | 3 |
| M2 | 18 | | 2 |
| M3 | 16 | | 4 |
| M4 | 12 | | 8 |
| R1 | 11 | 4 | 5 |
| R2 | 20 | | 0 |
| R3 | 20 | | 0 |
| R4 | 19 | | 1 |
| R5 | 14 | 3 | 3 |
| R6 | 16 | 4 | 0 |
| | yes | partially | no |
| T1 | 17 | | 3 |
| T2 | 9 | 10 | 1 |

was hard to manage multiple versions of codes or to take threats into account during development.

About the issues of requirements analysis, our questionnaires showed positive results as shown in Table 3. One subject who answered "no" in R1 wrote a comment that it was difficult to avoid threats even if he could identify them. Thus we have to provide effective mechanism to avoid threats in the next step. From the result of R5 and R6, our subjects assumed that developers were more responsible for identifying requirements and threats than users. This assumption seems to be quite rational because developers have more knowledge than users. So our tool is dedicated for application developers or integrator rather than users.

Finally, we review evaluation of our tool by our subjects. From the results of T1 and T2 in Table 3, their evaluation was not bad. It seems to be easy for our subjects to learn our tool because less than three hours were spent to learn it and the result T1 tells us most of them could understand how to use it. Typical positive comments were as follows.

- By using color changes, it is easy to identify permissions that are used in a requirement or a threat.
- In the same way, it is easy to identify requirements and threats that depend on the same permission.
- Automatically generated minimal policy is useful.

However, there were following negative comments.

- It is inconvenient to make relationship between a requirement or a threat and permissions.
- Because each permission is separated from a class where the permission belongs, it is difficult to understand the role of the permission.

## 6. Related Work

We suppose information systems in this research use fine-grained software components such as functions and/or classes. If such kinds of reuse are widely accepted, variety of components selection largely spreads and markets of such components grows soundly. There are already many researches about component selection and acquisition [8], [14]. However, threats caused by components composition were rarely discussed. This research and the tool directly handle such issue and partially support users and software integrators to such threats.

Requirements elicitation using interview is costly in general and [3] argued that a method to make it efficient is required. By using our tool, requirements analysts can enumerate the potential threats thus the tool helps such analysts to ask and explore what should not occur in an information system to be. This function largely contributes to elicit requirements efficiently.

Threats in this paper are very similar to obstacles in KAOS [18]. Their difference is that threats do not have to obstruct existing requirements but obstacles are basically identified by obstructing existing requirements or goals. Thus, threats in this paper are 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 [16]. 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 [10] is also systematic approach, but it is specialized for the requirements analysis of intrusion detection systems. A system called SoftwarePot [13] 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.

Our research and tool focus on one application used by one user rather than an information system used in an organization (many users). Thus, our research does not and cannot handle multiple users and/or roles in an information system because the application we focused basically has only one role. Taking such multiple roles into account, modeling techniques such as [7] or [9] are required. With respect to the Java specification and implementation, we only focus on the so-called "code-centric style" now. Therefore, we do not mind "who runs/executes a function" now. Java system already has a mechanism called "user-centric style" in Java Authentication and Authorization Service (JAAS) framework, so we want to extend our tool by taking roles into account by using JAAS framework.

In [2], an organizational policy is handled but our research is about security policy for an application. Thus, discussion and results in both researches cannot be simply compared. We think an organizational policy is a sum or product set of policies of applications in the organization. Thus, defining each application policy will sometimes contribute to define organizational policy.

In contrast with other researches about security requirements, our work is too simple. However, this is one of its advantages because our tool can be easily and effectively applied in educational settings. As shown in the case study in the previous section, students could use our tool easily, and they could have real experiences of threat activation. In fact, the educational materials in the case study embedded real malicious codes such as stealing personal information,

and the codes were sometimes activated during the course. By facing such real threats, students could deeply understand the importance of identifying threats as well as requirements. Tools and/or methods of other researches seem to be too complex to use in educational settings. There were a few researches about requirements engineering education [19] [5], and there is no research about educational aspects of security requirements. As reported in the previous section, a lecture using our tool gave good effect to students with respect to understanding requirements and threats.

In our research and tool, decision making such as how to reach trade-offs and/or mitigate threats is out of scope. Existing research results such as WinWin [1], DDP [4] and/or [15] can cope with our tool.

## 7. Conclusion

In this paper, we introduce a tool called PORATAM. The tool supports users, software providers and/or integrators to identify requirements, threats and policy for a mobile code application. Because our tool handles Java mobile code applications, users of our tool can easily meet threats caused by such applications during their analysis. From the result of our case study, our tool contributed learners to understand the importance of threats as well as requirements. Our tool also contributed them to find significant threats.

Basically, threats can be caused by a combination of several permissions. Currently, users of our tool have to find such combinations manually. We will extend our tool to propose possible combinations. It is difficult to decide a combination causes threats or not, but it is not difficult to enumerate possible combinations by using the dependencies and flows of data. Such combinations can also suggest unidentified requirements of users. In addition, our tool has to support stopping the gaps between such a combination and the meaning of a requirement or a threat as discussed in 5.3. Because semantic processing of a requirement or a threat is very difficult, such gaps can be stopped by using the empirical knowledge like design patterns. Another plan is to provide comparison mechanism of alternative codes. Currently, users have to replace a code to another manually, but the tool should recommend alternatives. If such comparison mechanism is provided, we can also compare non-functional features such as costs or response of codes. Current version of our tool does not explicitly handle the priority among requirements and threats. The tool currently enables its user to put requirements and threats in order respectively, but the ordering is not used formally in our tool. We will extend our tool by utilizing such priority for e.g., deciding trade-offs.

## References


[1] B. Boehm, P. Grunbacher, and R. O. Briggs. Developing Groupware for Requirements Negotiation: Lessons Learned. *IEEE Software*, 18(3):46–55, May/Jun. 2001.

[2] T. D. Breaux and A. I. Anton. Analyzing Goal Semantics for Rights, Permissions, and Obligations. In *13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 177–188, 2005.

[3] T. Cohene and S. Easterbrook. Contextual Risk Analysis for Interview Design . In *13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 95–104, 2005.

[4] S. L. Cornford, M. S. Feather, J. C. Kelly, T. W. Larson, B. Sigal, and J. D. Kiper. Design and Development Assessment. In *Proceedings of the Tenth International Workshop on Software Specification and Design (IWSSD'00)*, pages 105–114, 2000.

[5] C. Coulin et al. GONDOLA: An Interactive Computer Game-Based Teaching and Learning Environment for Requirements Engineering. In *REFSQ'04*, pages 113–126, 2004.

[6] R. Crook, D. Ince, L. Lin, and B. Nuseibeh. Security Requirements Engineering: When Anti-requirements Hit the Fan. In *IEEE Joint International Requirements Engineering Conference, RE'02*, pages 203–205, Sep. 2002.

[7] R. Crook, D. Ince, and B. Nuseibeh. On Modelling Access Policies: Relating Roles to their Organisational Context. In *13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 157–166, 2005.

[8] X. Franch et al. Using Quality Models in Software Package Selection. *Software*, 20(1):34–33, Jan./Feb. 2003.

[9] P. Giorgini et al. Modeling Security Requirements Through Ownership, Permission and Delegation. In *13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 167–176, 2005.

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

[11] H. Kaiya, K. Sasaki, and K. Kaijiri. A Method to Develop Feasible Requirements for Java Mobile Code Application. *IEICE Trans. Inf. and Syst.*, E87-D(4):811–821, Apr. 2004.

[12] H. Kaiya, K. Sasaki, Y. Maebashi, and K. Kaijiri. 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, Sep. 2003.

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

[14] S. Lauesen. COTS Tenders and Integration Requirements . In *12th IEEE International Requirements Engineering Conference (RE'04)*, pages 166–175, 2004.

[15] M. C. Robinson, S. E. Wallace, and D. C. Woodward. Risk Mitigation of Design Requirements Using a Probabilistic Analysis. In *13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 231–239, 2005.

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

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

[18] A. van Lamsweerde. Elaborating Security Requirements by Construction of Intentional Anti-Models. In *Proceedings of ICSE'04, 26th International Conference on Software Engineering*, pages 148–157, Edinburgh, May 2004.

[19] D. Zowghi and S. Paryani. Teaching Requirements Engineering through Role Playing: Lessons Learnt. In *12th IEEE International Requirements Engineering Conference (RE'04)*, pages 233–241, 2004.