Collaborative Policy Administration

Weili Han, *Member, IEEE,* Zheran Fang, Laurence T. Yang, *Member, IEEE,* Gang Pan, *Member, IEEE,* and Zhaohui Wu, *Senior Member, IEEE*

Abstract—Policy based management is a very effective method to protect sensitive information. However, the overclaim of privileges is widespread in emerging applications, including mobile applications and social network services, because the applications' users involved in policy administration have little knowledge of policy based management. The overclaim can be leveraged by malicious applications, then lead to serious privacy leakages and financial loss. To resolve this issue, this paper proposes a novel policy administration mechanism, referred to as Collaborative Policy Administration (CPA for short), to simplify the policy administration. In CPA, a policy administrator can refer to other similar policies to set up their own policies to protect privacy and other sensitive information. This paper formally defines CPA, and proposes its enforcement framework. Furthermore, in order to obtain similar policies more effectively, which is the key step of CPA, a text mining based similarity measure method is presented. We evaluate CPA with the data of Android applications, and demonstrate that the text mining based similarity measure method is more effective in obtaining similar policies than the previous category based method.

Index Terms—Collaborative Policy Administration, Policy Based Management, Mobile Applications, Android, Social Network Services

1 Introduction

The method of policy based management is widely used to manage complex and large scale network systems [1] [2] [3] [4]. The traditional framework of policy based management consists of four core components [5]: PDP (Policy Decision Point), PEP (Policy Enforcement Point), PAP (Policy Administration Point) and PR (Policy Repository). A well-trained policy administrator or group will specify, verify policies in PAP, and deploy the policies in PR. After a system runs, PDP will retrieve applicable policies from PR, and make decisions. PEP takes charge of the decision, such as satisfying the request where a subject wants to open a file (*authorization action*), or launching a logger to record system context (*obligation action*).

The overclaim of privileges, where a not well-trained administrator assigns more privileges than those are required of a subject, is a increasingly serious problem, especially when the method of policy based management is applied to emerging application scenarios, such as mobile applications [6] [7] [8] [9] and social network services [10]. For instance, during the process of Android application development, three roles are usually involved in the policy administration: *Application Developers* declare which permissions the application will request; *Application Marketers* verify whether the application is legitimate or not by an automatic tool; *Application Users* decide whether to approve the permission requests. These three roles are usually performed by those who are not well-trained in policy based management. That is, the developers usually declare more permissions than necessary because they are inclined to make the development of applications easier, or even misunderstand technical documents [9] [11]; the marketers usually tend to allow more applications regardless of the malicious permission requests; and the application users may not know what the requested permissions mean, thus approving all requests because they are eager to use the application. The same issue exists in social network services, where a user is asked to grant access to private data to third-party applications [12].

This challenge to policy administration is increasing serious due to the explosion of these applications. Among all smart phones shipped during the second quarter of 2012, Android OS smart phones had the largest global market share (68.1%) [13]. Furthermore, social network services have become one of the most popular web applications in the world [10]. For example, in April 2012, Twitter, an online microblogging service created in March 2006, announced that it commanded more than 140 million active users, and saw 340 million Tweets a day [14]. Although IETF proposed the protocol of OAuth 2.0 [15], the policy administration is still a serious problem in the policy based management of these emerging applications [12]. As a result, we should strengthen the policy administration mechanism in these application scenarios.

This paper proposes Collaborative Policy Administration (CPA for short). The essential idea of CPA is that applications with similar functionalities shall have similar policies which will be specified and deployed. Thus, to specify or verify policies, CPA will examine policies already specified by other similar applications and perform collaborative recommendation. The degree of similarity will be calculated by predefined

- W. Han and Z. Fang are with Software School, Fudan University, Shanghai, P. R. China 201203.
- L. Yang is with Department of Computer Science, St. Francis Xavier University.
- G. Pan and Z. Wu are with School of Computer Science and Technology, Zhejiang University, Hangzhou, P. R. China 310007.

algorithms, which cloud be a category based algorithm and a text mining based algorithm, etc.

The main contributions of this paper are as follows:

- We propose a novel method - collaborative policy administration, to help not well-trained users, even novices to specify and verify policies. We define the formal model of CPA. In this model, two main functions in policy administration are defined based on similarity measure methods, which will select similar policies as a refinement basis to assist administrators to design or verify their target policies.
- We propose a text mining based similarity measure method to help policy administrators to obtain similar policies. According to the evaluation, the proposed method is more effective than the category based similarity measure method, which is more widely used in other literatures [7] [12].
- We present an enforcement framework, and implement a prototype of CPA. The framework supports two types of user interfaces, and provides functions of collaborative policy design and collaborative policy verification.

The rest of the paper is organized as follows: Section 2 analyzes the trust model in mobile applications and social network services, then presents them as typical scenarios; Section 3 formally defines CPA; Section 4 introduces an enforcement framework; Section 5 evaluates the effectiveness of CPA by experiments; Section 6 discusses the rest key issues in CPA; Section 7 investigates the related work; and Section 8 concludes the paper and introduces our future work.

2 Background and Motivation

2.1 No Trusted Administration Point in Emerging Applications

In the traditional administration model [3], a professional expert or group will take charge of the policy administration, whose functions include policy design, policy verification, and policy deployment. When a system is being deployed, the expert or group will analyze what are the critical resources, then model them, and specify the management and security policies. These policies could be from the routines of organization management, or from the result of face-to-face discussions or meetings. After the policy design, some formal analysis tools could be applied to verify the policies. Even after the policy deployment, log-based analysis tools can help the expert or group to verify the designed policies: explore vulnerabilities, fix policies, and re-deploy them.

However, the emerging applications, especially mobile applications and social network services, challenge the existing trust model in the policy administration. In these emerging applications, common users, including software developers and end users, who do not possess the professional knowledge of policy based management, must specify or verify policies. The developer of a third-party application must request the privileges to be used by the application. When the developer writes the manifest files, he or she might overclaim privileges [9] [11]. The developer may not know what is at risk in detail if an application requests such privileges. As an end user of the third-party application, he or she must determine whether the requests for the privileges are legitimate. The task is very tough because the majority of end users do not know the risk of the approval in detail. Usually, the end user will approve all requests from third-party applications, because he or she wants to run the applications, thus falling into the traps of malicious applications.

As the second feature of the changing trust model, the involved parties in the policy administration could have the conflict of interest. A developer is inclined to request more privileges because this enables the developer to run his or her application with less privilege restrictions. However, an end user of the application is inclined to approve the fewest privileges possible to run the application, because more privileges expose more resources, thus lead to more risks, e.g., the leakage of privacy.

The third feature of the changing trust model is that a supervisor could verify the policy requests. The strengths of different supervisors are totally different. It depends on the strategies of the applications' distribution. For example, because the strength of Google Play Store is relatively weak, Android developers can publish many applications rapidly and easily. But Apple App Store deploys a stricter reviewing strategy. Thus the speed of application publishing in App Store is usually slow [16].

As a result of the changing trust model, overclaim of privileges is widespread in these emerging applications. This breaks the basic security principle: Principle of Least Privilege [17]. Although IETF proposed the open authorization protocol [15], it does not provide a policy administration model under the changing trust model. Thus, the changing trust model calls for more novel mechanisms to strengthen the policy administration.

2.2 Motivated Scenarios

2.2.1 Android Applications

In the Android security framework [6] [18] [19] [20] [21] [22], a developer rather than a professional policy administrator must set which permissions an application should request. And an end user of the application rather than a policy administrator must determine whether the requested permissions are legitimate for his or her mobile device. Due to the openness of the Android security framework, hundreds of millions of developers and users are involved. Without tools' support, the developers could not fully understand, or might even misunderstand the description of the requested permissions. As a result, overclaimed permissions are widespread in Android applications [11]. On the other hand, the end users usually approve all requested permissions due to the lack of knowledge of policy administration. The supervisors, application marketers, usually spend little effort on checking the applications. Thus, many applications with malicious behaviors can be uploaded to markets, and spread to lots of Android devices.

CPA will provide two functions to help application developers, marketers, and end users of Android: collaborative policy design and collaborative policy verification. Collaborative policy design assists the application developers in declaring permissions to achieve the applications' functionalities, observing the Principle of Least Privilege while ensuring the normal functionalities of the applications. On the other hand, collaborative policy verification helps the end users identify malicious permission requests, thus keeps the users from privacy leakage or financial losses. Moreover, for the marketers, collaborative policy verification mitigates the difficulty of filtering malicious applications.

2.2.2 Social Network Services

In social network services, third-party web-based applications could request sensitive information of end users. The problem is similar to the issue in the Android system. The differences between Android applications and social network services are as follows: First, the Android security framework only allows the end user to approve or deny all request permissions, but the social network services allow the end user to approve sensitive requests one by one. Second, the social network services can follow the protocol of OAuth (2.0), but the Android security framework is not standardized.

Literature [12] proposed several recommendation models for OAuth. This mechanism is similar to CPA. However, by leveraging CPA, the developer as well as the end user can benefit. The developer can determine which sensitive requests can be set according to other similar policies. As a result, he or she can develop securer and more acceptable applications for end users.

3 CPA: Definitions

The proposed collaborative policy administration includes two main stages: collaborative policy design and collaborative policy verification. We, therefore, formally define the CPA model as follows:

DEFINITION 1: **Collaborative Policy Administration Model**:

$$CPA := \{Admins, CPDM, CPVM\},$$

Here, $Admins$ refers to all involved policy administrators, including, e.g., developers, marketers, and end users in the Android framework.

We define *CPDM* as follows:

DEFINITION 2: **Collaborative Policy Design Model**:

$CPDM :=$
$\{PB_{history}, SimFunc, SUBJS, RefFunc, \Delta, PS_{ref}\}$

A policy administrator $\in Admins$ can obtain a refined policy set $\subseteq PS_{ref}$ according to a refinement function $\in RefFunc$, which is a refinement [3] driven by history data. Thus, it is very different from the traditional policy refinement methods [3], which are driven by a set of refinement rules or templates.

In DEFINITION 2, $PB_{history}$ refers to a policy base which contains a number of policies previously created both by administrator himself/herself and others. We abstractly define the content of $PB_{history}$ as follows:

$$PB_{history} := 2^{SUBJS \times PERMS}$$

Here, $SUBJS$ refers to the subjects in a system. E.g., in the Android system, all applications belong to $SUBJS$. $PERMS$ refers to all available permissions. E.g., there are 130 permissions in the Android system now [23].

Next, in DEFINITION 2, $SimFunc$ selects similar subjects, then output their policies according to the subject's attributes as the similar policies. E.g., when a subject has a *category* attribute, all other subjects in the *category* can be viewed as similar applications, e.g., 90% of *Minimal Acceptable Rate* in Appendix A.1 in the supplemental material. Formally,

$$SimFunc : SUBJS \times PB_{history} \rightarrow PS_{similar}$$

Here, $PS_{similar}$ refers to all policies of the similar subjects, as is described in the previous part.

In DEFINITION 2, $RefFunc$ refers to the refinement functions, each of which will output a policy set according to the attributes of a subject $\in SUBJS$, its similar policies $\in PS_{similar}$, and $\delta \in \Delta$, which may be a number, is a parameter of the function. Formally,

$$RefFunc : SUBJS \times PS_{similar} \times \Delta \rightarrow PS_{ref}$$

Here, PS_{ref} is a subset of $SUBJS \times PERMS$. Note that, each subject in the output PS_{ref} is equal to the input parameter of the subject $\in SUBJS$.

And we define *CPVM* as follows:

DEFINITION 3: **Collaborative Policy Verification Model**:

$CPVM :=$
$\{PB_{history}, SimFunc, SUBJS, VeriFunc, VeriResult\}$

A policy administrator $\in Admins$ can obtain a verification result $\in VeriResult$ for a target policy set $\in PS_{target}$, which contains all polices assigned to a target subject $\in SUBJS$, according to a verification function $\in VeriFunc$.

In DEFINITION 3, $SUBJS$ refers to the target subjects which will be verified. According to a target subject, we can obtain a target policy set PS_{target}, which is also a subset of $SUBJS \times PERMS$. Note that, there is only one subject in PS_{target}.

Finally, in DEFINITION 3, $VeriFunc$ refers to the verification functions, each of which will verify the target policy set, and provide a verification result. The result includes a simple conclusion $VeriResult$, e.g., whether the target policy set is suitable or not, or a vector of percentages, where each value represents how much percentage a permission in the target policy set occupies in the similar policies. We define $VeriFunc$ as follows:

$$VeriFunc : SUBJS \times PS_{similar} \rightarrow VeriResult$$

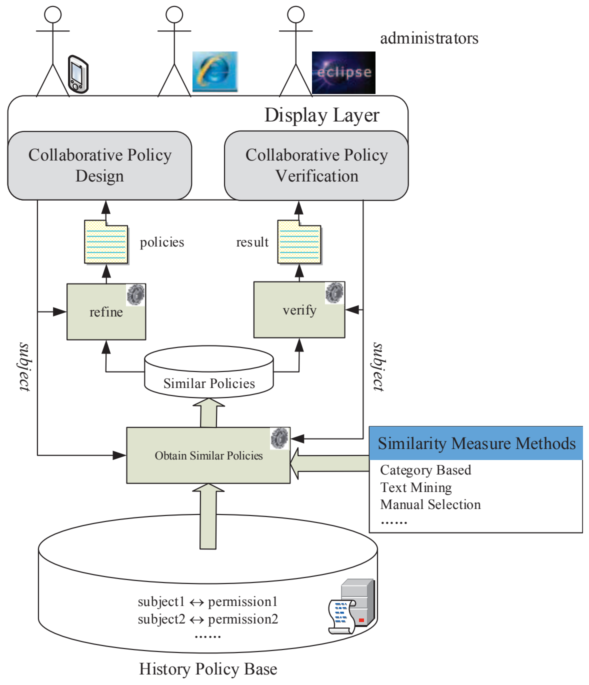


Fig. 1. Enforcement framework of CPA

4 CPA: Enforcement Framework

4.1 Overview

As is shown in Figure 1, a policy administrator can leverage the framework to administrate policies via a phone, web browser, or development tool. In Figure 1, the direction of arrows is the direction of key data flows. The history policy base and similarity measure methods are two key components in the enforcement framework. To enforce CPA, the administrator should prepare a sufficient number of policies at first.

Furthermore, collaborative policy design and collaborative policy verification are the two key functions provided by the framework, according to the definitions described in Section 3. These two functions depend on the history policy base and similarity measure methods. After obtaining the similar policies, the two functions call a refinement algorithm and a verification algorithm respectively.

Finally, collaborative policy design and collaborative policy verification will display the results to the administrator on various user interfaces, e.g., a phone, web browser, or development tool.

4.2 Key Algorithms

To enforce CPA, three key algorithms, similar policies algorithms, refinement algorithm, and verification algorithm, are proposed as follows:

4.2.1 *Similar Policies Algorithms*

Each similar policies algorithm obtains a similar policy set according to an input subject. For each policy in the **HPB**, each similar policies algorithm determines whether its subject is similar to the required subject. If so, add it to the similar policy set. There are many functions to judge the similarity such as checking the category of a subject, as described in Algorithm 1, or even friends' recommendation and manual selection.

Algorithm 1 Obtain Similar Policies Based on Category

```
Input:
  subject ∈ $SUBJS$
  HPB ∈ $PB_{history}$
Output:
  simpolicies ∈ $PS_{similar}$
  ————————————————————————————————
  for all policy ∈ HPB do
    if policy.subject.category == subject.category then
      simpolicies.add(policy)
    end if
  end for
```

The time complexity of Algorithm 1 is *O(n)*, where *n* refers to the number of policies in **HPB**. An index based on subjects in **HPB** can optimize the time complexity of Algorithm 1.

Furthermore, we propose a new method based on the text mining technique to obtain similar policy sets of an application in the Android framework. This new method leverages the description of a target application to find similar applications, then adds the requested permissions of the similar applications to the similar policy set of the target application. A TF-IDF [24] method is employed to create key words of application description, and scores will be generated according to the key words. Finally, the new method selects a pre-defined number (threshold) of applications according to the scores, and adds the selected applications' policy configurations to the similar policy set. Algorithm 2 shows our proposed similarity measure method in detail.

Algorithm 2 Obtain Similar Policies Based on Text Mining Method

```
Input:
  subject ∈ $SUBJS$
  HPB ∈ $PB_{history}$
Output:
  simpolicies ∈ $PS_{similar}$
  ————————————————————————————————
  initialize()
  query←parse(subject.description)
  for all subject ∈ HPB do
    doc←subject.description
    score← $a \times b \times \sum_{term \in query} (c \times d \times e \times f)(doc, term)$
    if score>simSubjs[simcountThreshold].score then
      simSubjs.removeLast()
      simSubjs.insertInDescendingOrderByScore(subject)
    end if
  end for
  for all subject ∈ simSubjs do
    simpolicies.add(subject.permissions)
  end for
  return simpolicies
```

In Algorithm 2, the *initialize* function involves declaring the **simpolicies**, assigning 0 to the score of every element in **simpolicies** and building the index for all application description from **HPB** if the index files are not available. The *parse* function tokenizes the description of the **subject** and returns a *query* object which is ready for searching. The statements inside the `for` loop are composed of a typical text mining procedure based on TF-IDF [24]. This procedure iterates on all subjects in the **HPB**. We execute the procedure with Apache Lucene [25]. a stands for *corrd(query, doc)*, which is the score factor based on how many of the *query* terms are found in the *doc*. b stands for *queryNorm(query)*, which is the normalizing factor used to make scores between queries comparable. For each *term* in the *query*, c stands for *tf(term∈doc)*, which is the *term*'s frequency in *doc*. d stands for *idf(term)*^{2}, which is the inverse document frequency of *term*. e stands for *term.getBoost()*, which is a search time boost of *term* in the *query*. f stands for *norm(term, doc)*, which is a few (indexing time) boost and length factors. *score* is calculated based on a, b, c, d, e, and f, as is shown in Algorithm 2. *simcountThreshold* refers to the threshold of similar subjects. When the *score* is higher than the score of the *simcountThreshold*th item in the similar subjects (*simSubjs*), the subject will be added into *simSubjs* in descending order by score. Finally, all policies of each subject in *simSubjs* will be added to **simpolicies**.

The time complexity of Algorithm 2 might be *O(n)*, where n refers to the number of subjects in **HPB**. However, each iteration of Algorithm 2 is heavy, because the algorithm leverages the algorithm of TF-IDF to generate the *score*s.

4.2.2 Refinement Algorithm

Algorithm 3 Collaborative Policy Refinement

```
Input:
  subject ∈ $SUBJS$
  simpolicies ∈ $PS_{similar}$
  $\delta \in \Delta$, it is a number here, e.g., 0.9
Output:
  refpolicies ∈ $PS_{ref}$
  ________________________________
  for all policy ∈ simpolicies do
    count[policy.permission]++
  end for
  for all permission ∈ PERMS do
    if count[permission]/simpolicies.size> $\delta$ then
      policy.subject ← subject
      policy.permission ← permission
      refpolicies.add(policy)
    end if
  end for
```

Algorithm 3 describes how to refine policies according to a parameter δ, which is a number here. The time complexity of Algorithm 3 is *O(n)*, where n refers to the number of policies in **simpolicies**.

4.2.3 Verification Algorithm

Algorithm 4 Collaborative Policy Verification

```
Input:
  subject ∈ $SUBJS$
  simpolicies ∈ $PS_{similar}$
Output:
  verires ∈ $VeriResult$
  ________________________________
  for all policy ∈ simpolicies do
    count[policy.permission]++
  end for
  targetpolicies ← ∀p∈ $HPB$: p.subject = subject
  for all tpolicy ∈ targetpolicies do
    verires[tpolicy.permission]←
        count[tpolicy.permission]/simpolicies.size
  end for
```

Algorithm 4 provides a quantified measure between the target policies and similar policies. Usually, the time complexity of Algorithm 4 is *O(n)*, where n refers to the size of **simpolicies**, because the size of **simpolicies** is usually larger than the size of *targetpolicies*, and the step to fetch *targetpolicies* can be optimized by using an index of subjects in **HPB**. The final output is a vector of percentages, each of which means how much percentage the permission of target policy occupies in the similar policies. To simplify the final result, we can design an aggregation algorithm to obtain a single number rather than a vector.

5 CPA: Evaluation

5.1 Permission Configuration Risk Index

To evaluate the effectiveness of CPA, we propose Permission Configuration Risk Index (*PCRI*), which can show how risky a measured application is. The essence of *PCRI* is that the more warning and dangerous highlighted permissions an application requests, the more risky the application is. The *PCRI* of an application is calculated based on the usage percentage and the critical level [7] of each permission the application requests. Here, only two levels, critical and non-critical, are used in *PCRI*. As a result, this proposed index can simplify the view of experiment results by integrating of the verification results.

During the calculation of *PCRI*, we first classify the requested permissions of an application by using warning percentage (P_w) and dangerous percentage (P_d), e.g., 80% and 50% respectively in Appendix A.1 in the supplemental material, which are preset by the policy administrators. If the usage percentage of a requested permission (P_p) is higher than the warning percentage, the permission is a *safe permission*; if the percentage of a requested permission falls between the warning percentage and dangerous percentage, the permission is a *warning permission*; if the percentage of a permission falls below the dangerous percentage, the permission is a *dangerous permission*.

TABLE 1
The calculation rules of PRI, $\vec{A}$ is a parameter vector of the following formulas

	Critical Permission	Non-critical Permission
Warning	$a_1 \times \frac{P_w - P_p}{P_w - P_d}$	$a_2 \times \frac{P_w - P_p}{P_w - P_d}$
Dangerous	$a_3 \times \frac{P_d - P_p}{P_d}$	$a_4 \times \frac{P_d - P_p}{P_d}$

Second, different types of permissions will contribute differently to the calculation of *PCRI*. After investigating the 130 currently available permissions in the security framework of Android [23], we argue that the security impact of abusing each permission varies. For example, an application which requests a rarely used the `READ_CONTACTS` permission is likely to be more harmful than another one using the `BATTERY_STATS` permission. The former one can collect the contacts' personal information which is very private, while the latter one is able to access only battery statistics. Sarma *et al.* identified 26 permissions as critical ones [7]. We, therefore, leverage them to calculate *PCRI*.

Third, we define Permission Risk Index (*PRI*) of each warning or dangerous permission, before we calculate *PCRI*. Note that, safe permissions have no contribution to the application's *PCRI*, because we use *PCRI* to help detect risky permission configurations.

For each warning or dangerous permission, we calculate its *PRI* by rules shown in Table 1. For example, for the application whose policy verification report is shown in Figure 6 in Appendix A.2 in the supplemental material, the *PRI* of the `WRITE_EXTERNAL_STORAGE` permission should be $a_1 \times \frac{80\% - 64\%}{80\% - 50\%}$, because it is a critical permission with a usage percentage lying in the range between the warning percentage and dangerous percentage.

Finally, after calculating the *PRI* of each requested permission of an application (we assume that an application *app* requests *n* permissions (*app.permissions*)), its *PCRI* will be obtained by the following formula:

$$PCRI(app) = \sum_{p \in app.permissions \wedge p.percentage < warning} PRI(p)$$

Here, *p.percentage* refers to the usage percentage of a permission *p*, and *warning* refers to the warning percentage defined by the policy administrator.

5.2 Effectiveness of Similarity Measure Methods

We evaluate whether the category based similarity measure method or our proposed text mining based one is more effective in obtaining similar policy sets. The evaluation is based on the two datasets mentioned in Appendix B in the supplemental material: one dataset contains 21,492 records of paid applications downloaded from the US Google Play Store; and the other dataset consists of 288 malicious application samples. And the measure methods are described in Algorithm 1 and 2. We use *PCRI* to evaluate the risk of applications in two datasets and calculate each application's *PCRI* according to the formula described in Section 5.1 with the two similarity measure methods and different parameters.

5.2.1 *Similarity Measure Methods for Android Applications*

Obtaining similar policy sets is a critical step in the CPA framework. As for Android applications, if the target policy set in policy verification is the permission configuration of a standalone game application, the obtained similar policy set should contain the requested permissions of other similar standalone game applications. From the characteristics of standalone games, we know that they should not have the right to access the Internet or send *short messages*. So the ideal similar policy set should not contain the `INTERNET` or `SEND_SMS` permissions. However, if some SMS (Short Message Service) applications are mistakenly judged as similar to standalone games, perhaps by wrong categorization, we will obtain a similar policy set which contains the `SEND_SMS` permission from those SMS applications and the results can be misleading, especially when we are verifying the permission configuration of a malicious standalone game application which sends premium rate messages without the acknowledging the phone's owner. When we verify the permission configuration of the aforementioned malicious application with similar policy set which contains the requested permissions of many SMS relevant applications, the verification result is bound to be very poor because we could see that the requested `SEND_SMS` permission is even marked in *green* rather than *red*. The poor result might mislead the application marketer into approving the sale of the malicious application.

Several similarity measure methods, e.g., category based, can be applied to obtain similar policy sets for Android application. On Google Play Store, applications are divided into 34 categories, e.g., *Books & Reference* and *Photography*. If we argue that applications belonging to the same category share some features or provide similar functionalities, one feasible method to obtain similar policy sets, therefore, is selecting permission configurations of all applications belonging to the same category as the subject. This attribute was also used in the related work of Shehab *et al.* [12]. Besides, we also propose the text mining based similarity measure method, as is shown in Algorithm 2.

The performance of similarity measure methods is the key factor of the two functions, collaborative policy design and collaborative policy verification, in CPA. We thus conduct the experiments to evaluate CPA according to the different similarity measure methods.

5.2.2 *Choose Threshold in the Proposed Method*

We first conduct experiments applying the text mining based method with different thresholds, $\vec{A} = \langle a_1, a_2, a_3, a_4 \rangle = \langle 3, 1, 10, 5 \rangle$, warning percentage=80%,

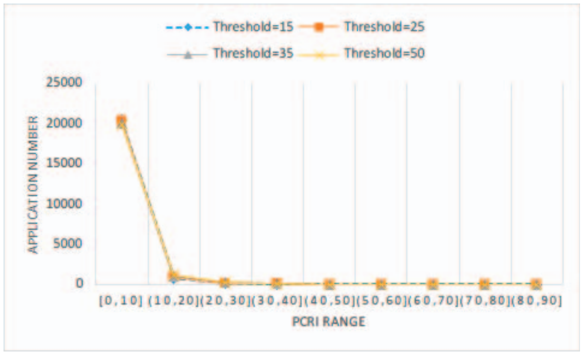


Fig. 2. *PCRI* distributions of the normal applications when we run the text mining based similarity measure method with different thresholds, $\vec{A} = \langle 3, 1, 10, 5 \rangle$, warning percentage=80%, and dangerous percentage=10%

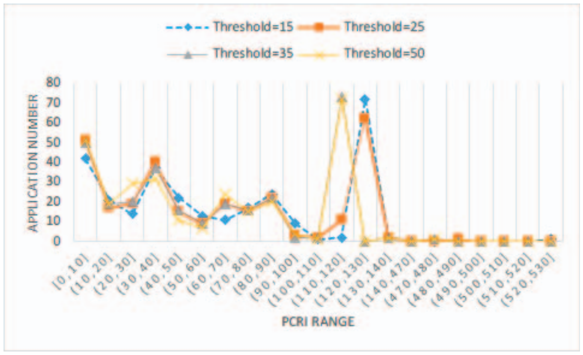


Fig. 3. *PCRI* distributions of the malicious applications when we run the text mining based similarity measure method with different thresholds, $\vec{A} = \langle 3, 1, 10, 5 \rangle$, warning percentage=80%, and dangerous percentage=10%

and dangerous percentage=10%. Figure 2 shows the effects of the different thresholds in Algorithms 2 for normal applications (the first dataset in Appendix B in the supplemental material). And Figure 3 shows the *PCRI* distributions of the different thresholds in Algorithms 2 for malicious applications (the second dataset in Appendix B in the supplemental material). We can draw two conclusions from Figure 2 and 3:

- *PCRI* can effectively figure out whether an application is normal or malicious. As is shown in Figure 2, 99.73%, 99.58%, 99.38% and 99.37% normal applications' *PCRI* is lower than 30 for each threshold respectively. As is shown in Figure 3, many more malicious applications have *PCRI* belonging to a higher *PCRI* range. For example, each threshold has about 70 malicious applications whose *PCRI* falls between 110 and 140. And only 26.74%, 30.21%, 30.90% and 34.38% malicious applications fall below 30 of *PCRI* for each threshold respectively.
- The threshold of *15* can be chosen as the suitable one in the text mining based similarity measure method. As is shown in Figure 2, the curves almost coincide with different thresholds. And in Figure 3, the curve of the threshold of 15 will provide more effective results than other thresholds (25, 35, 50), because the *PCRI* of malicious applications with threshold of 15 is the highest as a whole.

TABLE 2
Detection rates of the text mining based similarity measure method at different warning rates, with $\vec{A} = \langle 3, 1, 10, 5 \rangle$, warning percentage=80%, threshold=15, and different dangerous percentages

Warning Rate	Dangerous Percentage	Detected Malicious Count (288 in total)	Detection Rate
5%	10%	249	86.46%
	20%	238	82.64%
	50%	230	79.86%
10%	10%	257	89.24%
	20%	252	87.50%
	50%	248	86.11%
20%	10%	270	93.75%
	20%	269	93.40%
	50%	265	92.01%

5.2.3 Choose Dangerous Percentage in CPA

We conduct the experiments on different dangerous percentage (P_d) configurations. We employ two rates, *warning rate* and *detection rate* in the evaluation.

The *warning rate* decides the critical *PCRI* value. After sorting the applications in the normal application dataset in descending order by *PCRI*, we view the top *warning rate*, e.g., 5%, of applications as potentially malicious. The critical *PCRI* value is the largest *PCRI* value of the rest of the applications in the normal application dataset. For example, for text mining based similarity measure method with $\vec{A} = \langle 3, 1, 10, 5 \rangle$, warning percentage=80%, dangerous percentage=10%, threshold=15, if we set the *warning rate* as 5%, the critical *PCRI* value is 9.64, which is the largest *PCRI* value of the rest 95% normal applications.

The *detection rate* refers to the proportion of applications in the malicious application dataset whose *PCRI* is larger than the detection value.

Table 2 shows the *detection rate*s of the text mining based similarity measure method at different *warning rate*s, with warning percentage=80%, threshold=15 and different dangerous percentages.

As is shown in Table 2, with different *warning rate*s (5%, 10%, 20%), the *detection rate*s of the dangerous percentage=10% configuration are the highest among different dangerous percentage configurations. We can conclude from Table 2 that the dangerous percentage of 10% is the most suitable for the text mining based similarity measure method.

5.2.4 Compare the Proposed Method with the Category based Method

Table 3 shows the *warning rate*s and the corresponding *detection rate*s using the text mining based similarity mea-

TABLE 3
Detection rates of different similar measure methods at different warning rates, with $\vec{A} = \langle 3, 1, 10, 5\rangle$, warning percentage=80%, and dangerous percentage=10%

Warning Rate	**Method**	**Threshold**	**Detected Malicious Count (288 in total)**	**Detection Rate**
5%	Text Mining	15	249	86.46%
	Text Mining	25	236	81.94%
	Text Mining	35	234	81.25%
	Text Mining	50	231	80.21%
	Category based	NA	220	76.39%
10%	Text Mining	15	257	89.24%
	Text Mining	25	253	87.85%
	Text Mining	35	250	86.81%
	Text Mining	50	247	85.76%
	Category based	NA	232	80.56%
20%	Text Mining	15	270	93.75%
	Text Mining	25	265	92.01%
	Text Mining	35	263	91.32%
	Text Mining	50	259	89.93%
	Category based	NA	256	88.89%

*NA: Not Applicable

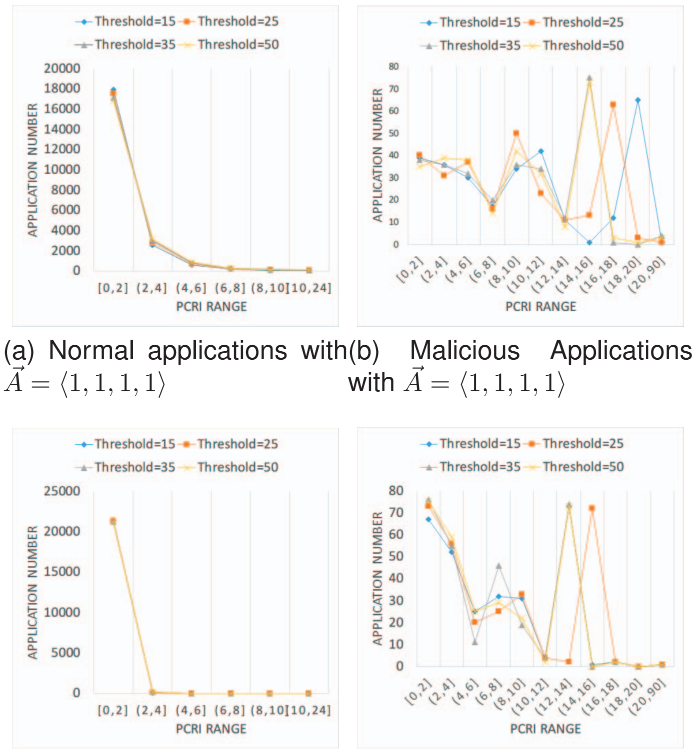
sure method with different thresholds and the category based similarity measure method.

Based on the data in Table 3, we can conclude that:

- The effectiveness of the text mining based method is better than that of the category based method. For all the *warning rate*s in Table 3, all *detection rate*s of the text mining based method are larger than that of the category based method.
- The threshold of *15* is proved to be suitable for similarity measure again. The *detection rate*s of the threshold of *15* with different *warning rate*s (5%, 10%, and 20%) are better than other similarity measure methods based on text mining. The possible reason is that fewer similar applications would be more aggregative than more similar applications according to applications' description.

We can see that the category based method perform worse than the text mining based method. The reasons are twofold: First, the category setting of Google Play Store is coarse-grained. The category information cannot fully represent the functionalities of applications. For example, networked and standalone games are both present in each game category and not separated. A typical difference between these two kinds of game applications, in terms of permission configuration, is that the former requires the `INTERNET` permission, while the latter does not.

Second, the description of an application can reveal more information about this application. For example, networked game applications usually come with key words such as *multiplayer*, *play with your friends* in their description. The text mining based method can derive key information from the application description and find applications with similar functionalities, which usually request similar permissions.



(a) Normal applications with $\vec{A} = \langle 1, 1, 1, 1\rangle$ (b) Malicious Applications with $\vec{A} = \langle 1, 1, 1, 1\rangle$

(c) Normal applications with $\vec{A} = \langle 0, 0, 1, 1\rangle$ (d) Malicious applications with $\vec{A} = \langle 0, 0, 1, 1\rangle$

Fig. 4. *PCRI* distributions of applications when we run the text mining based similarity measure method with different thresholds, different $\vec{A}$s, warning percentage=80%, and dangerous percentage=10%

5.2.5 Evaluate Different $\vec{A}$s

Finally, we conduct the experiments where $\vec{A}$ is set differently. The results of $\vec{A} = \langle 1, 1, 1, 1\rangle$ are shown in Figure 4.a and 4.b. And the results of $\vec{A} = \langle 0, 0, 1, 1\rangle$ are shown Figure 4.c and 4.d. According to Figure 4.a, 4.b, 4.c, 4.d, we can conclude that:

- The trends of curves are similar for all $\vec{A}$ ($\langle 3, 1, 10, 5\rangle$, $\langle 1, 1, 1, 1\rangle$, $\langle 0, 0, 1, 1\rangle$) for the normal applications. The difference is that the *PCRI* of $\langle 3, 1, 10, 5\rangle$ is the highest, and the $\langle 0, 0, 1, 1\rangle$ is the lowest.
- The significant difference happens for $\vec{A} = \langle 0, 0, 1, 1\rangle$ with the malicious applications. The *PCRI* of more applications in Figure 4.d fall in the area of small *PCRI*. This is bad for policy verification. Actually, the distributions of the malicious applications with $\vec{A} = \langle 1, 1, 1, 1\rangle$ are worse than the ones with $\vec{A} = \langle 3, 1, 10, 5\rangle$, if we compare Figure 4.b with Figure 3.

6 Discussions

6.1 Further Work on Policy Sets Obtained from Collaborative Policy Design

We can obtain a policy set from collaborative policy design. This policy set, however, may be an intermediate result and can be adjusted to meet the requirements of system management or security management. E.g., a developer of an Android application usually adjusts the policy set due to the extension of applications' functionalities. On the other hand, social network services might directly apply the policy set from the collaborative policy design, because users, especially who are friends involved in the services, tend to share more, and be less concerned with the security problem, thus the obtained similar policy set is very dense. As a result, the collaborative policy design should be accurate enough to be enforced. Thus, the refined policies can directly be used to determine whether the sharing operation should be allowed.

6.2 Vulnerabilities in CPA

A vulnerability of collusion exists in **HPB**. An attacker can put their malicious policy configurations into the base. Generally, only one attacker cannot affect the results of collaborative policy design and collaborative policy verification. However, if lots of attackers collude to input a large volume of malicious policy configurations, the **HPB** will be polluted and the similarity measure function could return a false result. Then the collaborative policy design and collaborative policy verification will show a wrong result to a policy designer or verifier. This vulnerability might be defended, if we carefully prepare **HPB**. Furthermore, the algorithms can still be robust in situations where only a few malicious policy configurations exist in the **HPB**.

The next vulnerability happens when an attacker maliciously abuse permissions with legitimate descriptions. For example, if the attacker releases a malicious application which sends premium rate SMSes without users' knowledge, but masquerades as a benign SMS application. SMS applications are usually justified to use the `SEND_SMS` permission. Then CPA could return a wrong verification result. The choices of description for malicious applications are limited. That is, if the attacker wants to maliciously leverage the `SEND_SMS` permission, he or she should camouflage the application as an SMS application rather than a game application which is more popular on the Google Play Store. However, this vulnerability is more severe, due to the coarse-grained permission model in the Android framework. To counter this threat, a finer-grained permission model with an improved administration mechanism should be developed. The proposed model must consider such burden of users' policy administration tasks. It is exactly an advantage of CPA to reduce the burden of policy administration.

7 Related Work

Policy administration is the key to protecting or operating information systems [3]. Only after a legitimate policy set is designed, can the systems run correctly. As a result, many researchers [26] [27] [28] proposed their works on this topic. This paper proposes a policy mechanism CPA under the current trust model, where a professional policy administrator or group is absent.

Recently, the security of the Android platform becomes a hotspot in the field of system security. Enck *et al.* [29] revealed that two-thirds of the 30 randomly-chosen popular third-party applications exhibited suspicious behaviors, such as disclosing sensitive information, and that half of the applications reported users' locations to remote advertising servers. Grace *et al.*'s analysis tool showed that among the 13 privileged permissions examined, 11 were leaked, with individual phone leaking up to 8 permissions, which can be exploited to wipe out the user data, send premium rate messages, etc. [30]. Ongtang *et al.* [31] proposed a complex policy model, and Nauman *et al.* [8] proposed runtime security constraint components for the Android platform. However, their goal is not to reduce the burden of policy design or policy verification. In fact, these two mechanisms may impose a higher requirement of professional knowledge on the Android security than previous methods. To assist an application user in determining whether he or she should accept the permissions, Nauman *et al.* [8] proposed to provide additional information in the installation interface. This tool, however, only shows the static description of permission in more detail. Enck *et al.* [32] proposed a tool named Kirin to analyze the manifest of Android application package files to ensure that the permissions requested by the application satisfy a certain safety policy.

To verify the overclaim of privileges among Android applications, Felt *et al.* [11] proposed a method where they detect API calls, then verify the policy configurations. However, their method cannot be effective in defending the threats from attackers who request a permission but implement a malicious relevant function. Sarma *et al.* [7] used the risk and benefit measures to verify the policy configurations. This paper, however, proposes two functions based on similar policies, collaborative policy design and collaborative policy verification. The proposed CPA and Sarma *et al.*'s method are different in mechanisms.

Policy recommendation [12] was proposed to simplify the policy administration in social network services. A user, as a verifier, can view the verification result of permission requests based on the similar policy sets. This paper proposes a text mining based method, thus can provide a more independent method to obtain similar policies, then improve the effectiveness of the functions of collaborative policy design and collaborative policy verification. Furthermore, CPA proposed in this paper also supports both collaborative policy design and collaborative policy verification, while the policy recommendation proposed by Shehab *et al.* [12] is only used in policy verification.

8 Conclusion and Future Work

This paper proposes a novel policy administration mechanism, CPA, to meet the requirements of the changing trust model, which has led to the widespread overclaim of privileges. CPA leverages the similar policies to design or verify a target policy set, then simplifies the policy administration for, especially, common users. This paper defines the formal model of CPA, and designs an enforcement framework. Furthermore, the paper proposes a text mining based similarity measure method to obtain similar policies. The evaluation by using a prototype and the data from Google Play Store shows the proposed text mining based measure method is more effective than the category based method which is usually used in previous related work.

In the future work, we will investigate the safety definition in CPA with a quantified method. Furthermore, we will improve the permission model with finer-grained access control for Android, especially, for `INTERNET` permission. Last but not least, we will strengthen the mathematics depth of the definitions and analysis of CPA.

Acknowledgments

The paper is extended from “Poster: Collaborative policy administration” [33] present at ACM CCS 2011. And this paper is supported by the 863 project (Grant NO: 2011AA100701), the project of Natural Science Foundation of Shanghai (Grant NO: 12ZR1402600). Thank Professor Wenyuan Xu and Mr. Hossen Mustafa for their helps and comments.

References

[1] A. K. Bandara, N. Damianou, E. C. Lupu, M. Sloman, and N. Dulay, *Handbook of Network and System Administration*. Elsevier, Nov 2007, ch. Policy Based Management.

[2] D. Verma, “Simplifying network administration using policy-based management,” *Network, IEEE*, vol. 16, no. 2, pp. 20–26, 2002.

[3] W. Han and C. Lei, “A survey on policy languages in network and security management,” *Computer Networks*, 2011.

[4] R. Yavatkar, D. Pendarakis, and R. Guerin, “A framework for policy-based admission control,” *RFC 2753*, no. 2753, 2000.

[5] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen, “Policy core information model – version 1 specification,” IETF, RFC 3060, Febrary 2001, http://www.ietf.org/rfc/rfc3060.

[6] W. Enck, M. Ongtang, and P. McDaniel, “Understanding android security,” *Security & Privacy, IEEE*, vol. 7, no. 1, pp. 50–57, 2009.

[7] B. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy, “Android permissions: a perspective combining risks and benefits,” in *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*. ACM, 2012, pp. 13–22.

[8] M. Nauman, S. Khan, and X. Zhang, “Apex: extending android permission model and enforcement with user-defined runtime constraints,” in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. ACM, 2010, pp. 328–332.

[9] D. Barrera, H. Kayacik, P. van Oorschot, and A. Somayaji, “A methodology for empirical analysis of permission-based security models and its application to android,” in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 73–84.

[10] G. Cheek and M. Shehab, “Policy-by-example for online social networks,” in *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*. ACM, 2012, pp. 23–32.

[11] A. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, “Android permissions demystified,” in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 627–638.

[12] M. Shehab and S. Marouf, “Recommendation models for open authorization,” *IEEE Transactions on Dependable and Secure Computing*, pp. 582–594, 2012.

[13] IDC, “Android and ios surge to new smartphone os record in second quarter, according to idc,” http://www.idc.com/getdoc.jsp?containerId=prUS23638712, 2012.

[14] Twitter, “Twitter turns six,” http://blog.twitter.com/2012/03/twitter-turns-six.html, 2012.

[15] OAuth, “The OAuth 2.0 protocol,” http://tools.ietf.org/html/draft-ietf-oauth-v2-22, 2011.

[16] A. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, “A survey of mobile malware in the wild,” in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2011, pp. 3–14.

[17] J. Saltzer and M. Schroeder, “The protection of information in computer systems,” *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.

[18] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, “These aren’t the droids you’re looking for: retrofitting android to protect data from imperious applications,” in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 639–652.

[19] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, “Google android: A comprehensive security assessment,” *Security & Privacy, IEEE*, vol. 8, no. 2, pp. 35–44, 2010.

[20] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, and S. Dolev, “Google android: A state-of-the-art review of security mechanisms,” *arXiv preprint arXiv:0912.5101*, 2009.

[21] K. Au, Y. Zhou, Z. Huang, and D. Lie, “Pscout: Analyzing the android permission specification,” in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012.

[22] H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy, “Using probabilistic generative models for ranking risks of android apps,” in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 241–252.

[23] Android, “Manifest.permission,” https://developer.android.com/reference/android/Manifest.permission.html.

[24] Wikipedia, “Tf*idf,” http://en.wikipedia.org/wiki/Tf*idf.

[25] Lucene, “Similarity (Lucene 3.6.1 API),” http://lucene.apache.org/core/3_6_1/api/core/org/apache/lucene/search/Similarity.html.

[26] R. Sandhu and Q. Munawer, “The arbac99 model for administration of roles,” in *Computer Security Applications Conference, 1999.(ACSAC’99) Proceedings. 15th Annual*. IEEE, 1999, pp. 229–238.

[27] N. Li and Z. Mao, “Administration in role-based access control,” in *Proceedings of the 2nd ACM symposium on Information, computer and communications security*. ACM, 2007, pp. 127–138.

[28] R. Bhatti, B. Shafiq, E. Bertino, A. Ghafoor, and J. Joshi, “X-gtrbac admin: A decentralized administration model for enterprise-wide access control,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 8, no. 4, pp. 388–423, 2005.

[29] W. Enck, P. Gilbert, B. Chun, L. Cox, J. Jung, P. McDaniel, and A. Sheth, “Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones,” in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, 2010, pp. 1–6.

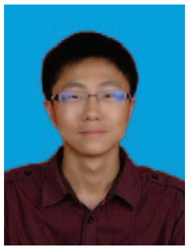
[30] M. Grace, Y. Zhou, Z. Wang, and X. Jiang, “Systematic detection of capability leaks in stock android smartphones,” in *Proceedings of the 19th Annual Symposium on Network and Distributed System Security*, 2012.

[31] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, “Semantically rich application-centric security in android,” in *2009 Annual Computer Security Applications Conference*. IEEE, 2009, pp. 340–349.

[32] W. Enck, M. Ongtang, and P. McDaniel, “On lightweight mobile phone application certification,” in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 235–245.

[33] W. Han, Z. Fang, W. Chen, W. Xu, and C. Lei, “Poster: collaborative policy administration,” in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 777–780.

Weili Han (M'08)is an associate professor at Fudan University. His research interests are mainly in the fields of policy based management, IoT security, information security, and distributed systems. He is now the members of the ACM, SIGSAC, IEEE, and CCF. He received his Ph.D. of Computer Science and Technology at Zhejiang University in 2003. Then, he joined the faculty of Software School at Fudan University. From 2008 to 2009, he visited Purdue University as a visiting professor funded by China Scholarship Council and Purdue. He serves in several leading conferences and journals as PC members, reviewers, and an associate editor.



Zheran Fang is an undergraduate student in software school at Fudan University now. He is currently a member of the Laboratory of Cryptography and Information Security, Fudan University. His research interests mainly include information security, policy based management.

Laurence Tianruo Yang is a professor in the Department of Computer Science at St. Francis Xavier University, Canada. His research interests include high-performance computing and networking, embedded systems, pervasive computing, and intelligent systems. Dr. Yang has a Ph.D. in computer science from the University of Victoria, Canada.

Gang Pan (S'04-M'05) received the B.Sc. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 1998 and 2004, respectively. He is currently a Professor with the College of Computer Science and Technology, Zhejiang University. He has published more than 100 refereed papers. He visited the University of California, Los Angeles, during 2007-2008. His research interests include pervasive computing, computer vision, and pattern recognition. Dr. Pan has served as a Program Committee Member for more than ten prestigious international conferences, such as IEEE ICCV and CVPR, and as a reviewer for various leading journals.

Zhaohui Wu (SM'05) received the Ph.D. degree in computer science from Zhejiang University, China, in 1993. From 1991 to 1993, he was a joint Ph.D. student with the German Research Center for Artificial Intelligence, Germany. He is currently a Professor with the Department of Computer Science, Zhejiang University. His research interests include distributed artificial intelligence, semantic grid, and ubiquitous computing. He is on the editorial boards of several journals and has served as chairs for various international conferences.