2013 IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing

Program Conversion for the Protection of Personal Information

Kuto, Kuniaki*, Takahashi, Kenichi*, Kawamura, Takao* and Sugahara, Kazunori*
* Department of Information and Electronics, Graduate School of Engineering
Tottori University, 4-101 Koyama Minami Tottori 680-8550, Japan
Email: {s082019, takahashi, kawamura, sugahara}@ike.tottori-u.ac.jp

***Abstract*—Some of Internet services request us to provide our personal information. When we use their services, we have to provide our personal information even if we cannot trust their service providers. This may cause the abuse of their personal information. Therefore, we propose a framework that prevents service providers from abusing users' personal information. In our framework, a user selects a method to use his/her information, and compels the service provider to use the method. Since the personal information is used through the method selected by the user, the user is able to prevent the service provider from the abuse of his/her personal information. Thus, the user can relief to provide his/her personal information to the service provider. We have some problems to be solved for the realization of our framework. In this paper, we discuss a way to install a method selected by a user into a program a service provider has.**

I. Introduction

The Internet services are already fundamental services in our daily and business life. For example, shopping sites, hotel reservation services and Internet auctions are popularly used. Their services request us to provide our personal information. However, when we provider our personal information once to the service provider, we cannot check anymore how our personal information is used by the service provider. Furthermore, a lot of cases such as infomation leakage, unauthorized use, phishing fraud and so on[1], [2], happen. Thus, we suffer for abusing of our personal information from service providers. We can avoid these cases when we should not provide our personal information to service providers. It, however, results in losing benefits to use the Internet services. Some of security techniques, such as authentication protocols, cryptographic algorithms, privacy policy and so on, are used for securing the Internet services. However, these techniques are applied by a service provider. Therefore, we cannot confirm the security techniques are really applied in the Internet services. Furthermore, even if we know more secure techniques, we cannot use their techniques. Thus, we wish a framework that enables to prevent service providers from abusing our pesonal information.

In this paper, we propose a framework that allows users to select a method for processing their personal information and compels the service provider to use the method. Our personal information is usually handled by a program that a service provider has. Therefore, our framework installs a method selected by a user into the program. Since the personal information is used through the method selected by the user, the user is able to prevent the service provider from abusing his/her personal information. As a result, the user can relief to provide personal information to the service provider. To realize our framework, we have some problems to be solved. In this paper, we discuss a way to install a method selected by a user into a program a service provider has.

The remainder of this paper is structured as follows. Section 2 describes the related studies. Section 3 presents the overview of our framework. After that, we we defines *protection policy* and *use policy*, which helps installing a method selected by a user into a program, at Section 4 and 5. Section 6 shows a way to install a method into the program. The paper is concluded in Section 7.

II. Related Studies

Cryptographic algorithms such as symmetric and public-key algorithms as well as techniques based on them such as digital signatures and public key infrastructure (PKI) have been proposed [3]. These algorithms and techniques aim at preventing message interception or identification of communication partners. Thus, they ensure message confidentiality, integrity, and availability of the message against malicious exploitation by third parties. Such techniques, however, do not prevent communication partners from abusing sensitive information released to them.

We often find a link on certain websites that points to the privacy policy adopted by that websites; this is termed as *Privacy Policy* on Yahoo!, *Privacy* on IBM, and so on. The privacy policy describes how the company treats personal information collected from users. The Platform for Private Preferences (P3P) [4] enables Web sites to express their privacy policies in a standard format that can be interpreted automatically by user agents. Thus, user agents can automate decision-making by comparing a company-defined privacy policy and user-specified privacy preferences. P3P, however, does not provide technical assurance that the sites will adhere to their respective privacy policies.

The Enterprise Privacy Authorization Language (EPAL) [5] provides fine-grained enterprise privacy policies, which employees within an organization are required to comply with. Whereas compliance to EPAL prevents the abuse of information by employees within an organization; it cannot provide an assurance to users that the methods used by the organization to manage their personal information are secure.

Various researchers have attempted to provide users with the right of information access or services based on trustworthiness [6], [7]. Their researches are aimed to developing trust relationships among users. However, it is difficult to define a general method for developing trust relationships.

978-0-7695-5088-6/13 $31.00 © 2013 IEEE
DOI 10.1109/HPCC.and.EUC.2013.225

IEEE computer society

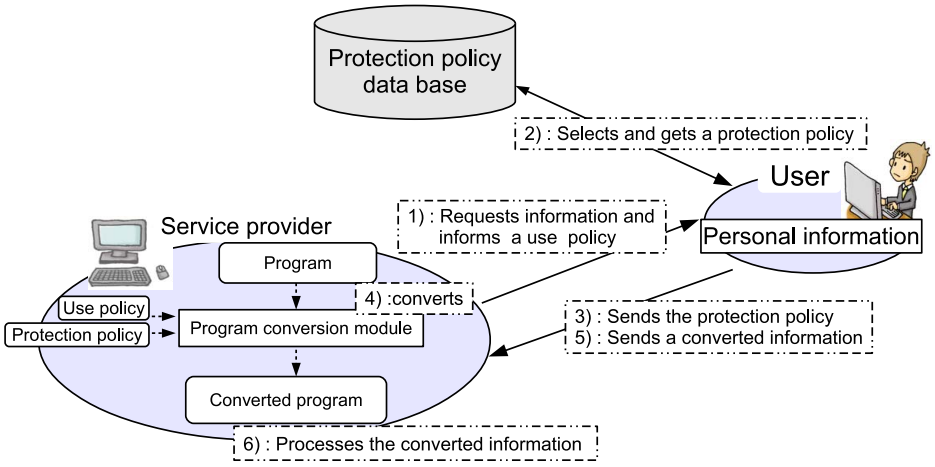

Fig. 1. Overview of the Proposed Framework

This is because trustworthiness depends on user conditions and/or situations. Moreover, trust relationships are not directly connected to the prevention of information abuse.

Inada et al [8] and Miyamoto et al [9] propose methods to preserve privacy by forbidding the disclosure of the combination of information that may lead to the loss of anonymity. However, these methods cannot preserve privacy nor protect information when the loss of only a single piece of information (e.g., credit card number or password) cause a problem.

III. Proposed Framework

We propose a framework that allows users to select a method in which their information is protected. A user prevents a service provider from abusing his/her information by that his/her information is processed through his/her selected method. In our framework, a user informs a method for processing his/her information to a service provider. The information of the method is defined in *protection policy*. *Protection policy* consists of installation manners of the method. We can know the manner for the installation of a method into a program of a service provider by the *protection policy*, however, the *protection policy* has no information about the program of a service provider. We cannot install a user's selected method without information about the program of a service provider. For example, we do not know, which variable deals with user's personal information, which function is applied to, in the program. Therefore, we introduce a *use policy* which defines above information. By getting a *use policy*, we can know how the personal information is processing in the program.

By getting a *use policy* and a *protection policy*, we can install the method into the program of a service provider. The installation of a method is done by a *program conversion module*. The *program conversion module* can know about the program of a service provider by reading a *use policy*, and manners to install a method a user selected by reading a *protection policy*. Figure 1 illustrates the overview of our framework.

1) A user requests a service to a service provider. Then, the service provider requests the user to provide a personal information. The user, however, worries about providing his/her personal information. Therefore, the user requests and receives a *use policy* from the service provider.
2) The user can know how his/her information is operated by the *use policy*. Then, the user accesses a *protection policy database* to select a method for dealing with his/her information he/she can relief, and get its *protection policy*.
3) The user informs the *protection policy* to the service provider.
4) The service provider converts the program which handles the user's information according to the *protection policy* and *use policy* by a *program conversion module*.
5) The user converts his/her personal information according to the *protection policy*, and provides the converted information to the service provider.
6) The service provider processes the converted information through converted program.

The personal information is converted at step 5, thus, the service provider is not easy to abuse it. On the other hand, the service provider can operate personal information through the program converted at step 4. Thus, it can prevent the service provider from abusing his/her personal information. We describe the details of *protection policy* in Section 4, *use policy* in Section 5, and *program conversion module* in Section

```
<INFORMATION> information </INFORMATION>
<OPERATION> operation </OPERATION>
<EXPLANATION>
   The explanation is written in natural language
</EXPLANATION>
<CONVERT-RULE>
   rules
</CONVERT-RULE>
```

Fig. 2. The Structure of Protection Policy

6.

IV. Protection Policy

The *protection policy* defines the installation manner of a method. A program is converted accroding to the *protection policy* for the installation of a method. The structure of the *protection policy* is shown in Figure 2.

A. Targetted Information and Operatrion

In our framework, a user selects a *protection policy* defining a his/her reliable method. It, however, may be troublesome for users because numerous protection policies are stored in the *protection policy database*. Therefore, we put <INFORMATION> and <OPERATION> to pick up appropriate protection policies from *protection policy database*. <INFORMATION> and <OPERATION> are written in machine-readble format. <INFORMATION> defines a subject that the *protection policy* enables to be applied. <OPERATION> defines an operation applied in the subject. The *protection policy database* picks up protection policies, which are possible to apply to a program, from numerous protection policies, and shows them to the user.

B. Explanation

There are several protection policies picked up by <INFORMATION> and <OPERATION> from *protection policy database*. When we select one *protection policy* from them, we have to understand how each *protection policies* handle his/her information. Such a information is defined in <EXPLANATION>. The value of <EXPLANATION> is written in natural language. Thus, we can select our reliable *protection policy* from them by reading <EXPLANATION>.

C. Program Conversion Rules

Our framework installs a method a user selects into the program of a service provider. For the installation of the method, we convert the program according to a *protection policy*. Therefore, we define rules to convert a program in <CONVERT-RULE>. <CONVERT-RULE> consists of <DATA-CONVERT-RULE>, <OPERATION-CONVERT-RULE> and <COMMUNICATION-RULE>. Figure 3 shows a brief description of these rules. <DATA-CONVERT-RULE> is a rule to convert *data* (a variable in a program). <OPERATION-CONVERT-RULE> is a rule to convert an operation. <COMMUNICATION-RULE> is a rule to control data transmission. A program is converted according to these rules for the installation of a method described in <EXPLANATION>.

```
<CONVERT-RULE>
   <DATA-CONVERT-RULE>
      converted-data <- method(data)
   </DATA-CONVERT-RULE>
   <OPERATION-CONVERT-RULE>
      new-method(converted-data) <- old-method(data)
   </OPERATION-CONVERT-RULE>
   <COMMUNICATION-RULE>
      data : denied
      converted-data : allowed
   </COMMUNICATION-RULE>
<CONVERT-RULE>
```

Fig. 3. Program Conversion Rules

<DATA-CONVERT-RULE>
Even if personal information a user wants to protect is defined in <INFORMATION>, we cannot protect the untouched personal information from a service provider. Therefore, we have to convert personal information to a form which enables to prevent a service provider from abusing of our personal information. Such a rule is defined in <DATA-CONVERT-RULE>. In a program, our personal information is dealt as a value in an variable. Therefore, <DATA-CONVERT-RULE> defines a rule to convert a value in an variable.

As shown in Figure 3, <DATA-CONVERT-RULE> is defined as "converted-data <- method(data)." Converted-data is a variable name for storing a value of data converted. Data is the name of a variable converted. Method is a method name for converting the value of data. For example, when <DATA-CONVERT-RULE> is defined as "h-pass <- hash(password)," the value of a variable named as password is converted by hash method and the converted value is stored in a variable h-pass.

<OPERATION-CONVERT-RULE>
Since personal information is converted according to <DATA-CONVERT-RULE>, an operation applied to personal information in an original program is useless anymore. Therefore, a rule is required to convert the operation to new operation which enables to process the converted personal information. Such a rule is defined in <OPERATION-CONVERT-RULE>.

<OPERATION-CONVERT-RULE> is defined as "new-method(converted-data) <- old-method(data)." converted-data and data are variable names. New-method is a method name which enables to process converted-data. Old-method is a method name which realizes the operation defined in <OPERATION>. For example, when <OPERATION-CONVERT-RULE> is defined as "equals(x, h-pass) <- equals(x, password)," a method equals(x, password) is converted to a method equals(x, h-pass).

<COMMUNICATION-RULE> Even if personal information is converted to other data, it is meaningless if a service provider can receive original personal information from a user. Also, the data converted by <DATA-CONVERT-RULE> should be sent to a service provider. Thus, we should be able to control the transmission of each data. Therefore, we define <COMMUNICATION-RULE>.

<COMMUNICATION-RULE> is defined as the pair of a

variable name and a permission. A permission is one of *allowed* and *denied*. If a permission is *allowed*, the value stored in its variable is allowed to be sent to a service provider; if *denied*, the value is denied to be sent. In the example in figure 3, the transmission code for the value in `data` takes away from a program and the code for `converted-data` are added appropriately.

A *program modification module* adds, deletes and replaces codes of a program according to <CONVERT-RULE>. As the result, a method a user selects is installed into a program a service provider has. We show more details of the installation steps in Section 6.

V. USE POLICY

A *protection policy* teaches a *program conversion module* a way to install a method, however, it has no information about a program the method is installed. For example, even if the *program conversion module* knows a way to convert personal information, it does not know which variable stores the personal information. Furthermore, we have to select a *protection policy* from a *protection policy database*, however we do not know which *protection policies* are possible to be applied in the program of a service provider. Therefore, we introduce a *use policy*. By getting a *use policy*, the *program conversion module* can know how the program handles the personal information. The *use policy* is composed of <INFORMATION>, <VARIABLE> and <OPERATION>. The structure of the *use policy* is shown in Figure 4.

```
<INFORMATION>information<INFORMATION>
<VARIABLE>
   <NAME>name</NAME>
   <TYPE>type</TYPE>
</VARIABLE>
<OPERATION>
   <FORMAT>
      <METHOD>(*<PARAMETER>)
   </FORMAT>
   <METHOD>
      <NAME>name</NAME>
      <RETURN>type</RETURN>
   </METHOD>
   <PARAMETER>
      <DATA>information</DATA>
      <NAME>name</NAME>
      <TYPE>type</TYPE>
   </PARAMETER>
</OPERATION>
```

Fig. 4. The Structure of Use Policy

<INFORMATION> defines personal information requested to a user. The user can know the subject of this *use policy* by <INFORMATION>. <VARIABLE> is composed of <NAME> and <TYPE> for indicating a variable storing personal information in the program. <NAME> defines a variable name, and <TYPE> defines its variable type. A *program conversion module* associates a *protection policy* with a *use policy* by comparing each <INFORMATION>, and knows which variable stores personal information.

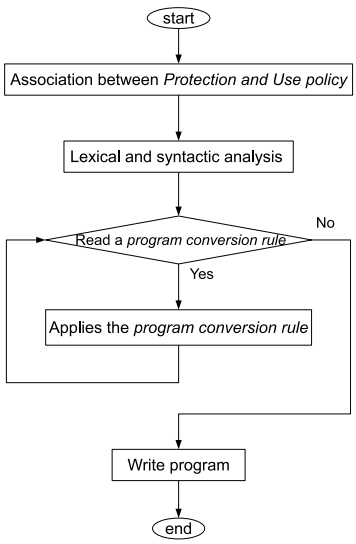

Fig. 6. Steps to Convert a Program

<OPERATION> is composed of <FORMAT>, <METHOD> and <PARAMETER>, that defines an operaion to be applied to <INFORMATION>. <FORMAT> shows which variables are used in the operation. <METHOD> is composed of a method name (<NAME>) and the type of return value (<RETURN>). <PARAMETER> is composed of <DATA>, <NAME> and <TYPE>, that define an argument applied to the operation. A user can know how his/her information is handled in the program by <OPERATION>.

VI. PROGRAM CONVERSION MODULE

A *program conversion module* installs a realiable method for a user into a program a service provider has according to a *protection* and a *use policy*. An example of program conversion is shown in Figure 5.

In this paper, we use Java as a program converted. In this example, the program receives a password from a user and does user authentication by the password. Therefore, the *use policy* has the definition of the password. From the *use policy*, *program conversion module* can know the password is stored in a variable "p", and used in an operation "equals(p, sp_p)." Here, a user worries about raw password to be used. Therefore, he/she selects an *protection policy* to encrypt a password [1]. The *protection policy* converts "password" to "en_password" by "encrypt" method; "en_password" to "org_password" by "decrypt"; "authentication(password, sp_password)" to "authentication(org_password, sp_password)." "password" and "org_password" are forbidded to sent out, but "en_password" is allowed. A *program conversion module* converts a program according to these rule for the encryption of a password. We show the steps to convert a program in Figure 6.

[1]In this example, the user regards the encryption is safe even if the encryption does not protect a password from a service provider.

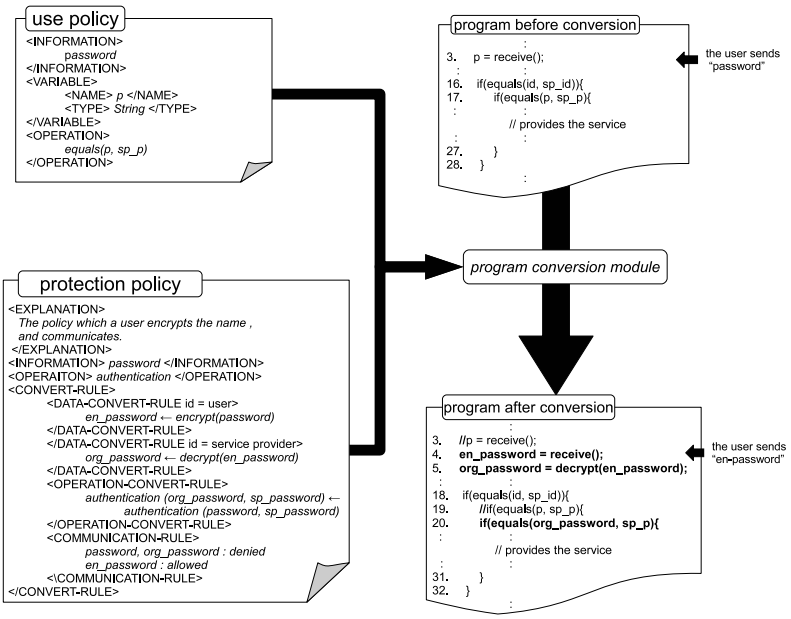

Fig. 5. An Example of Program Conversion

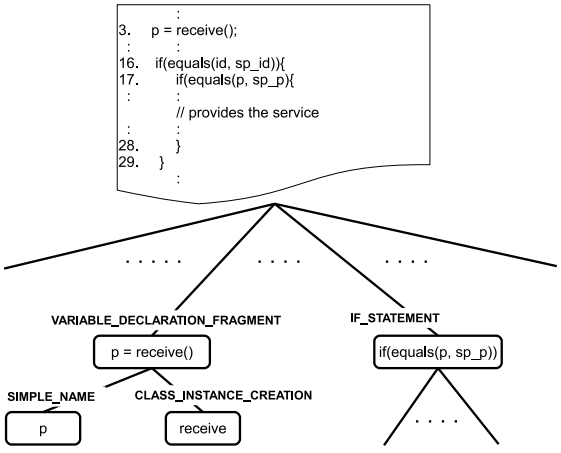

Fig. 7. Structured Program

A. Association between Protection and Use Policy

At first, a *program modification program* associates a *protection policy* with a *use policy*. A personal information a user wants to protect is written in <INFORNATION> of the *protection policy*, and a variable storing it in a program is written in <VARIABLE> of the *use policy*. Furthermore, an operation is defined in both policies. Thus, the *program modification program* associates them. As the result, it knows "password" is stored in "p" in the program, and the operation for "authentication" is "equals(p, sp_p).".

Next, we notice "sp_password" is used in "authentication(org_password, sp_password)." Here, "org_password" is defined in a *protection policy* but "sp_password" is not defined. Therefore, we have to find a variable storing "sp_password". When we look at <OPERATION> of the *protection policy*, we notice "sp_password" is used in "authentication(password, sp_password)." Since "authentication(password, sp_password)" is same with "equals(p, sp_p)," we can associates "sp_password" with "sp_p." In the same fashion, finally, a *program modification program* associates all data appeared in the *protection policy* with variables in the program.

B. Lexical and Syntactic Analysis

In this step, the program is analyzed for the preparation of program conversion. We use *ASTParser*[10] to analyze a program. ASTParser creates a abstract syntax tree to express the structure of a program. The example of the structured program is illustrated in Figure 7.

As shown in Figure 7, the program is separated in each line; each line is decomposed into some parts. For example, the thrid line of the program which handles "p" is decomposed into "p" (SIMPLE_NAME) and "receive" (CLASS_INSTANCE_CREATION). We can find a variable and a method from this tree by calling method prepared for each. For example, if we want to find a variable "p" from the program, we can find where "p" appears in by calling "SimpleName.getIdentifier()" which returns the value of SIMPLE_NAME. Thus, we can know, where each variables are appeared in, where each operations are used in, when our personal information is substituted in a variable, and understand the flow of each variables.

C. Applies Program Conversion Rules

The *program conversion module* converts a program according to <DATA-CONVERT-RULE>, <OPERATION-CONVERT-RULE> and <COMMUNICATION-RULE>.

At first, the *program conversion module* tries to convert the operation "equals(p, sp_p)" written in the *use policy*. <OPERATION-CONVERT-RULE> in the *protection policy* is "authentication(org_password, sp_password) <- authentication(password, sp_password)." Since "password" is stored in a variable "p", "sp_password" in "sp_p", we can knows "authentication(password, sp_password)" is "equals(p, sp_p)." Therefore, the *program conversion module* finds "equals(p, sp_p)" from the program; replaces it to "equals(org_password, sp_p)."

Here, "org_password" is not appeared in the program, therefore, it tries to create "org_password." We can know "org_password" is created by "org_password <- decrypt(en_password)" from <DATA-CONVERT-RULE>. Therefore, the *program conversion module* inserts codes to create "org_password" before "equals(org_password, sp_p)." Also, "en_password" is not appeared in the program. However, we can know "en_password" is received from a user because the <COMMUNICATION-RULE> of "en_password" is *allowed*. Therefore, the *program conversion module* inserts codes to receive "en_password." Furthermore, since the <COMMUNICATION-RULE> of "password" is *denied*, codes to receive "password" is removed. In similar manner, the user stops to send "password"; creates "en_password"; and sends "en_password" to the service provider.

As the result, the *program conversion module* can create the program handling personal information as an encrypted data. After that, the *program conversion module* writes the resulted program into Java formatted file. The service provider compiles the converted program, and processes user's personal information through the converted program. Thus, the user can be relieved from providing his/her personal information to the service provider.

VII. Conclusion

In this paper, we discussed a way to install a method selected by a user into a program a service provider has for realizing the framework that enable to prevent service providers from abusing our pesonal information. To realize our framework, we define a *protection policy* that defines the installation way for information protection, and a *use policy* that defines how personal information is handled in the program of a service provider. A *program conversion modules* converts the program of a service provider according to the *protection* and the *use policy*. We used techniques of lexical analysis and syntactic analysis for the program conversion. Since our framework compels the service provider to use through the converted program, the user would not worry about providing his/her personal information to a service provider.

The future works will include the evaluation of our framework and the extention of the *program conversion module* to cover more complicated programs.

Acknowledgment

This research has been supported by a Grant-in-Aid for Young Scientists (B), 23700098.

References

[1] NPO Japan Network Security Association,Information Security Incident Survey Report 2012, http://www.jnsa.org/result/incident/2012.html

[2] Council of Anti-Phishing Japan, A Monthly Report, http://www.antiphishing.jp/report/monthly/

[3] D.R. Stinson, editor., Cryptography: Theory and Practice, Crc Pr I Llc, 1995.

[4] P3P project, http://www.w3.org/P3P

[5] The EPAL 1.1, http://www.zurich.ibm.com/security/enterpriseprivacy/epal/

[6] G. Theodorakopoulos, J. Baras, Trust Evaluation in Ad-Hoc Networks, WiSe04, pp. 1-10, 2004.

[7] D. Xiu, Z. Liu, A Dynamic Trust Model for Pervasive Computing Environments, FTDCS 2004, pp. 80-85, 2004.

[8] M. Imada, K. Takasugi, M. Ohta, K. Koyanagi, LooM: A Loosely Managed Privacy Protection Method for Ubiquitous Networking Environments, IEICE Trans. on Comm., Vol.J88-B, No.3, pp. 563-573, 2005.

[9] T. Miyamoto, T. Takeuchi, T. Okuda, K. Harumoto, Y. Ariyoshi, S. Shimojo A Proposal for Profile Control Mechanism Considering Privacy and Quality of Personalization Services, DEWS 2005, 6A-o1, 2005.

[10] Exploring Eclipse's ASTPaerser, http://www.ibm.com/developerworks/opensource/library/os-ast/