

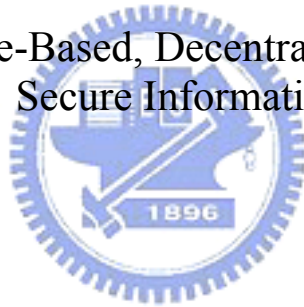
國立交通大學

資訊科學系

碩士論文

一個以規則為基礎之非集中式資料
共享安全機制

A Rule-Based, Decentralized Approach to
Secure Information Sharing



研究生：徐嘉宏

指導教授：陳俊穎 博士

中華民國九十四年七月

一個以規則為基礎之非集中式資料共享安全機制

A Rule-Based, Decentralized Approach to
Secure Information Sharing

研究生：徐嘉宏

Student：Cha-Hun Hsu

指導教授：陳俊穎

Advisor：Jing-Ying Chen

國立交通大學
資訊科學系
碩士論文



Submitted to Institute of Computer and Information Science
College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer and Information Science

July 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年七月

一個以規則為基礎之非集中式資料共享安全機制

學生：徐嘉宏

指導教授：陳俊穎博士

國立交通大學資訊科學系

摘要

隨著網際網路的普及而成為大眾資訊交流的媒介，資訊共享的安全與隱私權保護的課題益發重要。保護機制不完善的資料共享系統可能會導致機密的資料洩漏。為了減低這樣的風險，已有許多不同資訊安全的機制與技術被提出。而這些方法大多採用以標籤為基礎的存取控制機制。進一步地說，這些安全機制為系統所管理的資料及使用者貼上適當的標籤，並根據這些標籤來制定及執行不同的安全政策。然而，為了讓安全政策可以正確的運行，這些方法需要複雜且冗長的管理。另一方面，大多數的方法主要是針對資料的存取，而對於處理資料的工具本身並沒有進一步的控制。由於判定一個的工具能否存取某筆資料取決於使用此工具的使用者，使得安全管理的正確性不易控制。在本篇論文中，我們提出一個資料分享的模型，能提供一個有彈性且非集中式的方法來控制資訊流動，而不須仰賴集中式的管理。此模型中，每個使用者可以指定其個人的存取等級，並根據此存取等級對資料及工具貼上標籤。並且，藉由非集中式的工具降級設定，每個使用者可以個別地增強或減弱某工具所能存取的資料等級，進而達到更方便且安全的政策制定。最後，為了減少對每筆資料貼上標籤的麻煩，我們的模型使用了一個以規則為基礎的標籤機制來處理不同等級的資料。總結來說，我們的模型利用靈活的策略管理，統合且改進了現存的模型，能作為未來網際網路資訊共享安全機制的核心。

A Rule-Based, Decentralized Approach to Secure Information Sharing
student : Cha-Hun Hsu Advisors : Dr. Jing-Ying Chen

Department of Computer and Information Science
National Chiao Tung University

Abstract

As the Internet becomes a ubiquitous environment where people with different background and purposes can share their information, information security and privacy have become an increasingly critical issue. With poor protection, information systems may leak sensitive information to the open public. To reduce such risks, there have been many security mechanisms and technologies proposed and developed. Most of these approaches rely on a mixture of label-based access control and information flow mechanisms. Specifically, they enforce various security policies by attaching suitable labels to information as well as users and grant data access based on these labels. However, many labeling approaches often require complicated managerial efforts in order to set up and enforce security policies correctly. In addition, they focus primarily on labeling data without paying too much attention to sharing tools that process these data. As a result, whether a given tool can access a piece of data depends on who is invoking the tool, making correct security management more challenging. In this thesis, we propose an information sharing model that permits controlling information flow in a flexible, decentralized manner, where each user can specify his/her own access hierarchy instead of relying on centralized security management. Based on the access hierarchy, each user can label both data and tools in a consistent manner, and can realize data declassification by restricting or relaxing the access level of individual tools. Finally, to reduce the overhead associated with individual data labeling, we introduce a rule-based labeling mechanism to associate data with their access levels correspondingly. In summary, we believe our model is a unification of and

improvement over existing access control mechanisms, and can contribute to secure information sharing over the Internet.



致謝

這篇論文的完成，我想要感謝的人很多很多。首先是要感謝實驗室的同學、學弟、以及已經畢業的學長們。林建宏學長、郭訓宏學長、張舜禹學長、劉許吉學長，在我的研究生涯中總是不吝給我指導與協助。和我有革命情感的同學們，詹亦秋、林君翰、莊景棠、鄧嘉源，陪我一同度過充實研究生涯。

我也要感謝給我的口試委員曾文貴教授以及王伯堯研究員，在口試的時候給我許多有用的看法以及建議，讓我的論文能夠更加完整。

此外，我要感謝我的爸爸、媽媽、哥哥，他們是我的精神支柱，總是在我背後默默支持我，讓我有走完這兩年研究生涯的動力。我也要感謝我的女朋友淑如，在我困頓落寞的時候，總是給我適時的鼓勵，讓我有勇氣與自信能夠完成這篇論文。

最後，我想要感謝我的指導教授陳俊穎老師，在這兩年的日子總是耐心的給我勉勵與指導，除了學習到的專業知識，我想我在研究生涯中最讓感到充實的，就是學習何謂作研究應有的態度。我想，對於傳授我這些知識與觀念的指導教授，言語以及文字的感謝已經太膚淺了。

List of Contents

摘要	i
Abstract.....	ii
致謝	iv
List of Contents	vi
List of Figures.....	vii
Chapter 1 Introduction.....	1
Chapter 2 Background.....	6
2.1 Internet as an Information Sharing Environment	6
2.2 Security and Access Control Mechanisms	7
2.3 Information Flow Mechanisms.....	8
2.4 Role-Based Access Control.....	11
2.5 Decentralized Information Flow Control	12
2.6 Declassification	13
2.7 Security Policies and Policy Standards	13
Chapter 3 Motivation and Objective	15
3.1 A Medical Data Analysis Example	15
3.2 A Bank Example.....	17
Chapter 4 A Rule-Based Decentralized Information Flow Approach.....	20
4.1 Overview	20
4.2 Object and Principal Labeling.....	21
4.3 A Rule-Based Labeling Approach.....	23
4.4 Tool Sharing and Declassification.....	24
Chapter 5 Type Systems Approach to Information Flow.....	27
5.1 The Volpano Approach	27
5.2 Type System of Our Model	32
Chapter 6 Comparisons with the ML Model.....	37
Chapter 7 System Design and Implementation	43
Chapter 8 Related Work	47
Chapter 9 Conclusions.....	49

List of Figures

Figure 1. Illustration of a general information sharing system	2
Figure 2. Access control mechanisms restrict rights for a subject to access object	7
Figure 3. Illustration of an access control matrix, where the row represents subjects, the column objects, and each cell corresponds to the privileges that subject has over the object.	8
Figure 4. Information flows from data object A to data object B and from data object A to data object C.	9
Figure 5. Lattice-based information flow policies.	10
Figure 6. Definition of the can-flows relation from L_1 to L_2	12
Figure 7. Definition the join operator $L_1 \cup L_2$	12
Figure 8. A medical data analysis scenario where statistical analysis is conducted on patient records.	16
Figure 9. The bank example where each customer keeps asset information private from other customers and the bank wants to keep its total asset private.	18
Figure 10. Overview of our information sharing model, that consists of a set of principals, data, and tools.	20
Figure 11. An object label contains sub-labels each indicating the security level of the object from each principal's perspective.	22
Figure 12. Definition of the label structure.	22
Figure 13. Definition of the can-flows relation.	22
Figure 14. Definition of the join operator among labels.	23
Figure 15. Our model allows a principal to attach individual labels on data in the realm.	24
Figure 16. Each user can specify declassification on each tool.	25
Figure 17. Typing rules for the Volpano model.	30
Figure 18. Evaluation rules for the Volpano model.	31
Figure 19. Language syntax definition.	32
Figure 20. Typing rules for our model.	33
Figure 21. Flow declaration and un-declaration for typing rule (letvar)	34
Figure 22. Evaluation rules in our model.	36
Figure 23. Checking password example which is a procedure to check whether the username and password is correct.	37
Figure 24. Constraints of the password checking example presented by decentralized label model.	38
Figure 25. The constraint solution of password checking example.	38
Figure 26. Checking password example performed by our model.	39
Figure 27. Type inference and constraints generation of tool password checker using type inference algorithm.	40
Figure 28. Type inference and constraints generation of tool string matcher using our type inference algorithm.	40
Figure 29. Overview of our system implementation.	43
Figure 30. The flow path of data processing.	45
Figure 31. The flow path of policy specification.	45
Figure 32. The flow path of declassification on a tool.	46

Chapter 1 Introduction

Today, people rely extensively on the Internet for information exchange, professionally or socially. Many Internet applications such as WWW, e-mail, and instant messengers all provide people with different means to exchange information and collaborate with each other. As a result, the issue of security and privacy protection is becoming more critical. For example, if an information sharing system contains spyware, information may be stolen or destructed in an undesired way. The Center for Democracy Technology [CDT], a non-profit organization dedicated to promoting civil liberties values on the Internet, also summarizes potential security risks resulting from inadvertent sharing of sensitive personal information via spyware or adware. [Dav 03] also list some potential security risks.

To simplify discussion, in this thesis we are concerned with information sharing systems in general. As depicted in Figure 1, an information sharing system consists of data elements, tools that create and operate on these data, and users that establish connections with the system to gain access to the data and tools inside. Information sharing is achieved when data created by one user can be accessed by another. Note that information sharing systems need not maintain data or tools in a centralized place. Many important Internet applications can be regarded as information sharing systems. For example, the most popular and successful one, i.e. WWW, conceptually forms a global information space that contains numerous Web pages interconnected through hyperlinks, and people use Web browsers to access Web pages created by others behind Web servers.

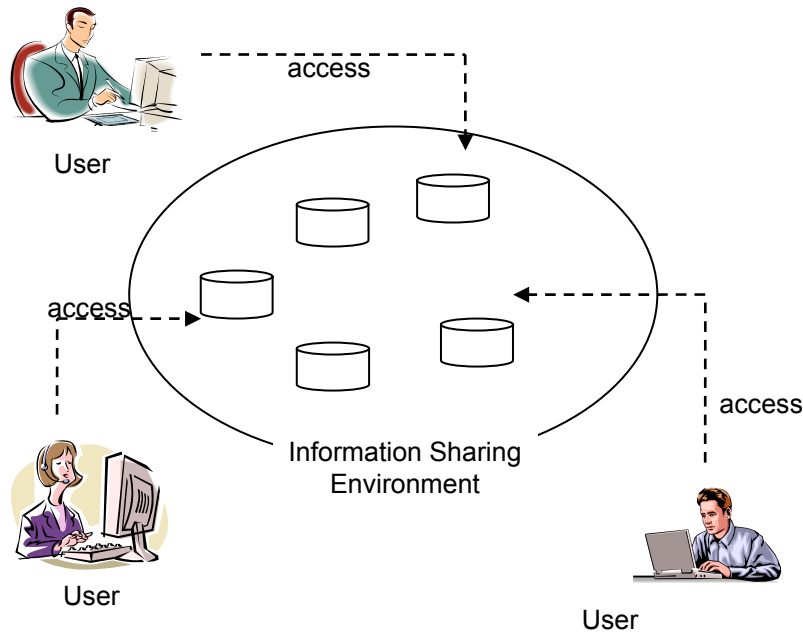


Figure 1. Illustration of a general information sharing system

Information security concerns the control of use and dissemination of information within an information sharing system. Privacy protection is the ability of a user to control the availability of information about oneself, and is becoming an important topic of information security nowadays. Security policies are security and/or privacy requirements that an information system needs to enforce. Some systems allow administrators or individual users to define security policies in terms of rules about different conditions under which certain data can be access by a particular user. Policy management is a general term that includes policy specification, deployment, and reason over policies, updating and maintain policies and enforcement. For example, Internet Explorer allows user to configure security and privacy settings. Security setting is done through configuring the degree of trust for various web sites in terms of download permissions and execution permissions such as blocking advertisements from specific sites. Privacy protection is done through setting the permissions for particular web sites to obtain sensitive information about the use through cookies.

Enforcing security policies within a complex system to prevent careless information leakage is nontrivial. There has been numerous security mechanisms developed over the last decades to address this challenge. Common security mechanisms include authenti-

cation, authorization, discretionary access control, and information flow. Authentication is the process of verifying whether a person is really someone he/she claims to be, while authorization is the process of checking whether that person is permitted to access a given system or resource. Further introduction to authentication and authorization is beyond the scope of this thesis.

Discretionary access control (DAC) is a kind of security mechanisms that prevents destruction of information by defining or restricting the rights of subjects (users, sometimes also called principals) as well as processes that act for them to access a particular data object. One main objective of DAC is to protect information from accident or undesired destruction. The mechanism provided by Internet Explorer as described above is an example of DAC. In general, DAC-based systems grant access rights based on data and subject labeling. For example, most operating systems provide DAC mechanisms that allow users to specify the access rights for individual directories and files they own, such as read, write, and execute operations, for different classes of users within the system. However, most DAC based systems support labeling an inefficient way, especially when the volume of data is large. Consider, for example, the strategies and amount of work needed to properly set access rights for all the files a user own in an operating system, to make sure no accidental leak of information occurs from time to time.

A more serious problem of DAC is that it does not control the dissemination of information, that is, how a user uses the data objects after he/she gains access to them, including improper propagation of the information to other people. To prevent such information leakage, *information flows*, also called *mandatory access control* (MAC), is an approach to preventing unauthorized propagation of information. With data objects properly classified, MAC mechanisms ensure that high level information can not flow to low level user, and users are greatly relieved from otherwise tedious policy management.

Despite the advantage of MAC over DAC where information flow can be controlled more rigorously, such restriction also prevents user from releasing part of his/her information to other users or tools for further analysis – an important requirement for many information systems. To remedy this problem, certain declassification mechanisms are proposed to “breach” security requirements in an acceptable degree. For example, the "acts-for" relation proposed in [Mye97] allows one principle to grant all access right to another principal, which also implies some trust relation between the two. That is, if

subject A acts for subject B, subject A can get all information that principal B can get, even to change principal B's policy temporarily to facilitate certain data processing tasks principal A needs. However, as with most DAC mechanisms, the acts-for approach to declassification relies heavily on mutual trust among principals, thus careful design and management of the acts-for relations are required.

Take a medical information system as an example, where a doctor may own certain diagnosis information that a statistician wants for further statistical analysis. To achieve this goal, the doctor can declassify such sensitive data by permitting the statistician to act for him, assuming certain agreement or mutual understanding has been established between the two, so that the statistician can get access to the diagnosis data. Obviously, once the act-for permission is granted, the doctor has no explicit control about how the statistician can evaluate the data. Moreover, even if the statistician has no bad intention, he may leak some sensitive information by accident, or when he/she under security attack unknowingly.

The discussion above about permitting one subject to act on behave of another, either through the acts-for mechanisms in [Mye97] or through other comparable mechanisms, brings up the issue of controlling privacy policies. Many systems, including both DAC and MAC-based systems, often rely on centralized security administration governing the protection and closure about private data. Individuals often have limited options about how their personal data can be protected and used. To deal with this problem, [Mye97] also proposes *decentralized* information flow model, so that different users can specify their own policies, respectively, yet the overall information flow is under control.

When the complexity of policy management is concerned, MAC also faces the problem of correctly managing security policies. It is easy to prevent improper dissemination of information in small systems. In large system, however, controlling information flow is difficult, especially when the data, tools, and users involved are diverse and the relations among them become complicated.

In this thesis we propose a decentralized information flow model that attempts to address security and privacy issues highlighted above. Following Myers and Liskov's work, we focus on mechanisms supporting decentralized policy management in a general information sharing environment. Specifically, in our model, users define their own information flow lattice structures, respectively, and associate data with security levels

drawn from such lattice structures using user-defined rules. Tools that process and generate data are also an integral part of the model, to enable true secure information sharing.

Normally, tools respect the information flow jointly defined by the users. When declassification is needed, our model does not rely on the acts-for relation. Instead, tools are labeled individually in a decentralized manner similar to data labeling, but with downgrading specification describing the tolerable range for each user using the same user-defined lattice. With this approach, users can grant different tools with different capacities based on their understanding of the input-output characteristics of the tools, thereby eliminating the dependency on mutual trust among users. To further reduce potential security risks, we also define a simple yet flexible programming language such that for tools written in the language, automated analysis and verification can be performed to give users definite security guarantee about these tools.

We have developed a prototype system implementing our ideas using Prolog, and described the system using type theoretical approach. Specifically, our system accepts a user query or processing request only when no security breach can happen. The rest of this thesis is organized as follows. In chapter 2, we first present some important background on security mechanisms such as access control and information flow models, and the decentralized approach to information flows pioneered by Myers and Liskov. In chapter 3, we use two information sharing examples to help present the motivation behind our research. In chapter 4, we describe our decentralized information flow model in detail, followed by some analysis of the model using type systems in chapter 5. In chapter 6, we differentiate our model from the work by Myers and Liskov and explain how to achieve similar information sharing policies using our model. Finally we give some discussion about related work and conclude with some future research directions.

Chapter 2 Background

2.1 Internet as an Information Sharing Environment

As the Internet rose during mid-1990, new forms of digital information exchange such as e-mail, FTP, and WWW emerged and gradually transform how people work and collaborate with each other ([Jap00, Gaw03, Xu04]). A typical information worker spends over one and half an hour on e-mails each day in average - 20% of the working hours. Employees get 50% to 75% of their relevant information directly from other people, and more than 80% of enterprises reside in hard drives. Thus we can see that information sharing plays an important role in modern human life.

However, using public media such as the Internet for information exchange and sharing among people also created many problems and pose challenges for companies. Among the most important issues is security and privacy protection. As pointed out by the Center for Democracy and Technology ([CDT]), potential security risks include

- Inadvertent sharing of sensitive personal information – users may leak sensitive personal information by inadvertent mistake.
- Spyware and adware – software may collect user's personal information to third party without user's knowledge.
- Security concern – information exchange also introduces the risks similar to those faced by Internet users generally, where people should be careful to only execute tools whose source they trust, and they should safeguard their computers when online.

To ensure security and privacy of information, one needs to prevent sensitive information from leaking out inadvertently. Privacy and security are inextricably linked; both are need to establish a complete protection mechanism for an information sharing system, since one can not ensure privacy without suitable security control.

2.2 Security and Access Control Mechanisms

Information security concerns the protection of data against unauthorized access. Information security mechanisms are ways to deal with the particular access requests, which contain authentication, authorization, and access control (Figure 2). Authentication is the process of checking whether a person or software agent is permitted to access a particular resource. Typical authentication involves password checking, although other methods such as fingerprinting or other biometrics are also common nowadays.

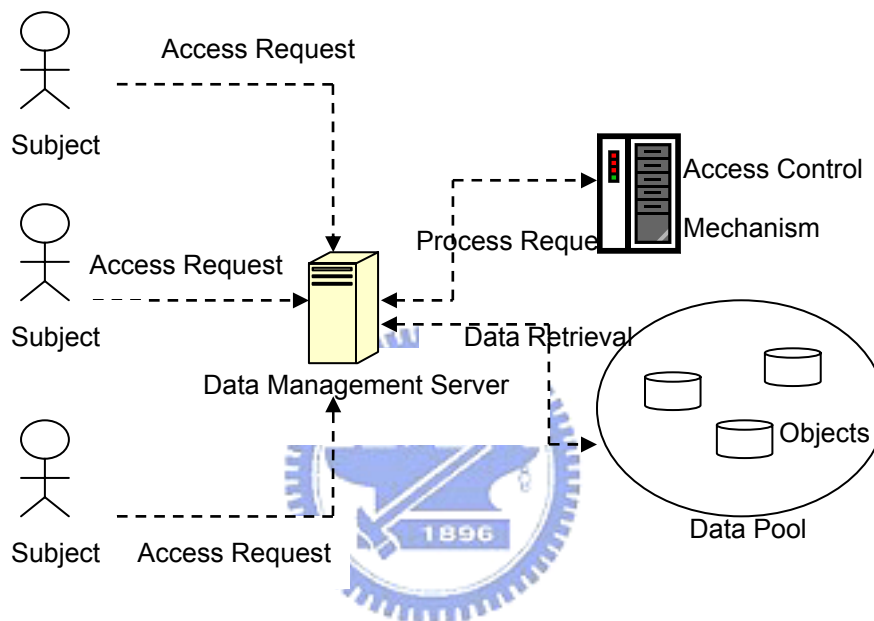


Figure 2. Access control mechanisms restrict rights for a subject to access object

Authentication is part of the more general access control topic that concerns the protection of computer resources against undesired access. Basic access control concepts include *objects*, *subjects* and *access rights*. Objects are a general concept that refers to any computer resources providing information, ranging from files, to memory and other IO devices, to complete network accessible systems. Subjects include both users and tools that access objects on behalf of the users who use them. Access rights describe which operations (e.g. read, write, and execute operations) are permitted for a subject to perform on a given object. Since early 1960s, access control has been a major research topic for information systems such as database management systems and operation systems. Modern access control systems can grant or deny access requests from a particular

subject according a wide variety of criteria, such as time, current location, or domain of the subject.

Popular access control mechanisms include *discretionary access control* (DAC) and *mandatory access control* (MAC). DAC was originated from the academic area. In a DAC model, each object belongs to an owner, who can restrict or grant the access request from other users. One of the most commonly used approaches to DAC is access control matrices [San92]. Figure 3 shows an example of access control matrix in which each row represents a subject and each column represents an object. Each cell in the matrix contains the privileges the corresponding subject has over the corresponding object. The access control matrix may change when modification to the subjects, objects, and/or privileges in each cells are made.

Object → Subject ↓	John's File	Bob's File	Mary's File	Ada's File
John	rwX			r
Bob		rwX		
Mary	r		rwX	
Ada	rw	rw		rwX

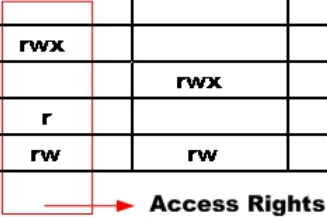

Access Rights

Figure 3. Illustration of an access control matrix, where the row represents subjects, the column objects, and each cell corresponds to the privileges that subject has over the object.

DAC mechanisms are commonly used in multi-user operating systems. However, access control mechanisms control immediate access request without taking into account information flow path implied by a given, outstanding collection of access rights [Den76]. Thus they can not prevent dissemination of objects [Li03] [Mye97] [Zan04]. Also, DAC lacks a theoretical base to ascertain that objects are protected without any mistakes made by users, and careful specification and management of policies is needed.

2.3 Information Flow Mechanisms

MAC mechanisms, also called *information flow* models, are another approach to information security, with the objective to prevent unauthorized information flow among

objects and subjects. As shown in Figure 4, information is said to flow from object A to object B whenever information associated with object A affects the content of which associated with B. Information also flows between objects and subjects (e.g. from B to C in the figure). Information flow is transitive, thus in Figure 4 an information flow also occurs from object A to subject C.

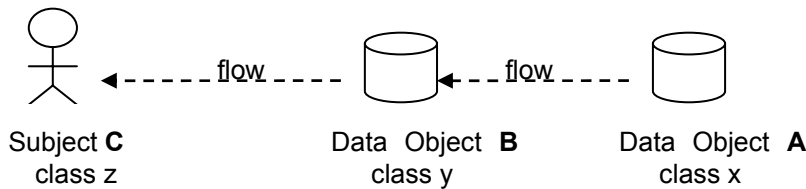


Figure 4. Information flows from data object A to data object B and from data object A to data object C.

Secure information flow is usually achieved by assigning each object and subject a *security class*, or *security label*, where possible security classes are drawn from an access hierarchy, or more generally, a lattice structure. Since security classes have ordering relations among each other reflecting their relative sensitiveness or secrecy, MAC mechanisms ensure that information cannot flow from higher security classes to lower ones. As also shown in Figure 4, object A, B and subject C are assigned security classes x, y, and z, respectively, and the information flow among A, B, and C implies that x, y, and z are in an ascending order. Note that it is usually assumed that labels on data objects and subjects would not change, and the assumption is known as tranquility [San93].

The most widely known information flow models are the Bell-LaPadula model [Blp75] which deals with information confidentiality, and the Biba model [Bib77] that deals with information integrity. Confidentiality is concerned with disclosure of information, i.e. preventing information from flowing downwards from higher security label to lower one. Information integrity, on the other hand, is concerned with the modification of information. For example, in a bank information system, confidentiality is concerned with preventing clients from finding other clients' cash balances, while integrity is concerned with preventing the clients from changing the balances.

The Bell-LaPadula model defines two properties: the *simple security property* and the *star property*. The simple security property prescribes the so-called *no-read up* princi-

ple, that is, a subject can not read an object higher than its own security class. The star property prescribes the *no-write down* principle, that is, a subject can not write to an object lower than its own security level. The basic idea of the Biba model is similar to Bell-LaPadula model, which also defines two similar properties from the integrity perspective: the simple integrity property and the integrity star property. The simple integrity defines no-write up, that is, a subject can not write to an object higher than its own integrity level. The integrity star property defines no read down, that is, a subject can not read an object higher than its own integrity level.

Formally, an information flow model is defined as a set of security classes, a binary relation between security classes, and join operator on security classes. For example, the information flow model by Denning [Den76] is defined as a triple $\langle SC, \rightarrow, \oplus \rangle$, where SC is a set of security classes, $\rightarrow \subseteq SC \times SC$ is a binary relation (denoted as "can flow" relation) on SC , and $\oplus: SC \times SC \rightarrow SC$ is a binary class-combining or join operator on SC . Under the assumptions below, the information flow model forms a lattice structure:

1. The set of security classes SC is finite
2. The can-flow relation \rightarrow is a partial order on SC .
3. SC has a lower bound with respect to \rightarrow .
4. The join operator \oplus is totally defined least upper bound operator.

Figure 5 shows an example lattice-based information flow policy that says information can only flow from low level objects to high level objects, but not the other way around.

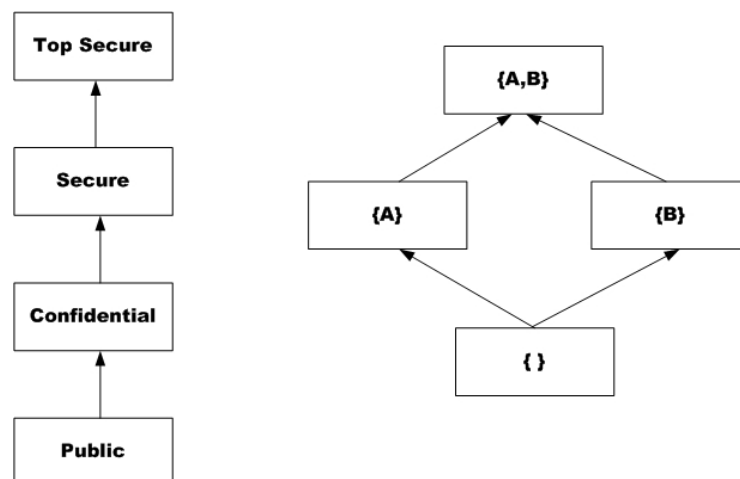


Figure 5. Lattice-based information flow policies.

Information flow models provide more precise control of information than DAC models. In DAC, an object becomes accessible to some subject when appropriate permission is specified in the access control matrix. But once the subject gets access to the object, it can propagate the content of the object in an arbitrary way. Information flow models realizes the so called non-interference [Gog82] property, which states that low level output should not be affected by high level input. This assures that a low level agent cannot get information about the high level inputs. With non-interference property, information sharing system can prevent low level subjects to observe change of high level subjects.

2.4 Role-Based Access Control

One problem DAC-based systems encounter in practice is when access control policies change frequently. As stated in [Clk87], earlier access control models could not satisfy the practical requirements needed by commercial organizations. It is realized that large voluminous data in organization are not owned by individual users but the organization itself. In addition, individual users may have various access rights under different circumstances. These considerations inspired the so called Role-Based Access Control (RBAC) proposed in 1988, when Lochovsky and Woo defined roles and organized them into a hierarchy [LW88].

The basic idea of role-based access control is to define roles as the entities and authorize them rather than authorize subjects directly. In real environments, a user can play different roles under different contexts. Instead of granting access to an object of a subject, the authorization is represented as a role's access to an object, and the subjects can be assigned to different roles. RBAC models often form relation of roles as a hierarchy reflecting different positions in an enterprise, and support more dynamic policy changing and relax the otherwise rigid constraints.

Although role-based access control supports more flexibility than DAC, there are some scenarios can not be supports well by role-based access control. An example is that a team has a goal and member work collaboratively. It is hard to express real world policy by role manipulations [Bel04]. More importantly, information flow addressed by MAC is also not addressed by RBAC.

2.5 Decentralized Information Flow Control

Traditional information flow models form a multi-level security system, which is useful for centralized administration. For example, the Biba model and the Bell-LaPadula model originated from a centralized, military setting. However, these models are not suitable for commercial environment where each user has his/her own security policy on information sharing.

Myers and Liskov [Mye97] developed a decentralized information flow model, which we refer to as *the ML model* from here on, permits setting information flow policy in a decentralized manner. In the ML model, a label of an object contains an owner set, whereas each owner in the owner set maintains a set of readers. The set of valid readers to the object is defined as the intersection of all the reader sets. For example, a label L can be described as $\{O_1: R_1, R_2; O_2: R_2, R_3\}$ where O_1, O_2, R_1, R_2, R_3 are subjects. The owner set of L , denoted as $owners(L)$, is $\{O_1, O_2\}$. The reader set of O_1 in L , denoted as $readers(L, O_1)$, is $\{R_1, R_2\}$ and the reader set of O_2 in L is $\{R_2, R_3\}$. The valid readers of L is defined as the intersection of $\{R_1, R_2\}$ and $\{R_2, R_3\}$, namely, $\{R_2\}$. The can-flows relation between labels L_1 to L_2 , i.e. whether L_1 can flow to L_2 , is shown in Figure 6, which indicates that L_1 should contain fewer owner and more readers than L_2 .

$$\begin{aligned} &owners(L_1) \subseteq owners(L_2), \\ &\forall O \in owners(L_1), readers(L_1, O) \supseteq readers(L_2, O). \end{aligned}$$

Figure 6. Definition of the can-flows relation from L_1 to L_2

The join operator of two values Label of $L_1 \cup L_2$, as shown in Figure 7, is defined as the join of readers and union of owners. The upper bound is data which is owned by every principals and no reader allowed, and the lower bound corresponds to data can flow anywhere. We can see that information can flow from more readers and fewer owners to fewer readers and more owners, and a global lattice is formed.

$$\begin{aligned} owners(L_1 \cup L_2) &= owners(L_1) \cup owners(L_2), \\ readers(L_1 \cup L_2, O) &= readers(L_1, O) \cap readers(L_2, O). \end{aligned}$$

Figure 7. Definition the join operator $L_1 \cup L_2$

In addition to the definition of the lattice structure, another important component of the decentralized model is the constraint that a subject has the right to modify his/her own reader set in the label for a given object without influencing other users' policies. This decentralized policy specification, compared with centralized downgrading approaches, are more suitable for secure information sharing in a complex, distributed environment.

2.6 Declassification

During information processing in mandatory information flow models, the resulting object labels become increasingly restrictive and make the information less available for subjects. Sometimes, the security levels of the objects need to be relaxed so that other parties can read it. This kind of label relaxation is called *declassification*. From the perspective of information flow control, declassification allows high level entities to flow to low level entities.

The ML model described in the previous section also introduces the *acts-for* relation indicating whether a subject subsumes all privileges of another subject. That is, if the subject A gets the right to act for the subject B, then A has all the access rights B possesses. The acts-for relation facilitates declassification due to the fact that once subject A acts for subject B, subject A can temporarily modify subject B's reader sets of the labels for all involving objects.

As [Sam00] states, however, the acts-for mechanism is powerful but dangerous, and users must be careful to allow only trusted principals to act on their behalf.

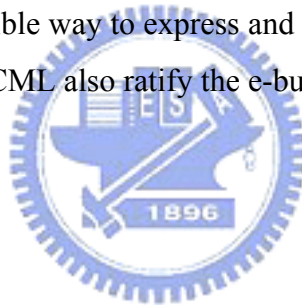
2.7 Security Policies and Policy Standards

Information security policy defines a set of rules for information availability within an information sharing system. How to specify and enforce security policy precisely is a long standing problem because managing policies involving sharing tools among subjects with diverse goals, which can involve complicated declassification management.

There are currently many standards supporting information security and privacy ([Appel], [PHI], [ASS], [XACML]). For example, HIPAA (The Health Insurance Portability and Accountability Act, [HIPAA]) established rules such as those in [PHI], [ASS] to protect the confidentiality and other personal health information. Another example is

Appel (A P3P Preference Exchange Language, [Appel]) for website privacy defined by P3P ([P3P]). P3P specifies an architecture comprising user agents, privacy reference files, and privacy policies. Appel allows webmasters to specify a standard set of multiple-choice questions, which result in tags embedded in the web site's home page. On the other hand, P3P-enabled Web browsers allow users also define their own privacy requirements, such as stating whether they allow their names disclosed to third parties. Together, P3P and Appel help Web sites announce their privacy practices while letting users automate their accepting and rejecting decisions. Note that P3P does not attempt to enforce or ensure privacy through technology—for example, by cryptographic or anonymization techniques. Instead, it relies on social and legal pressures to compel organizations to comply with their stated policies

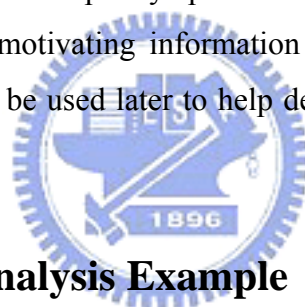
XACML (eXtensible Access Control Markup Language, [XACML]) is a policy language standard developed by OASIS (Organization for the Advancement of Structured Information Standards, [OASIS]), which aims at protecting content during enterprise data exchange by defining flexible way to express and enforce access control policies in a variety of environments. XACML also ratify the e-business standards.



Chapter 3 Motivation and Objective

Our goal is to develop an information sharing system that permits decentralized specification and management of information flow policies for individual users, yet reduces the burden of policy management as well as the risks of accidental information leakage. Earlier models do not meet these requirements, and several common problems still remain. They either handle declassification in an inefficient way, or fail to manage large volume of data properly, mainly due to the cost of manual data labeling.

In a medical data analysis system, for example, patients may want to share their own information with researchers to help advance medical research, yet they want to control the exposure of their private information according to their own policies, respectively, such as permitting the access to his diagnosis record only for certain statistical analyses. To better illustrate the decentralized policy specification and management problem, in this section we describe two motivating information sharing examples adopted from [Mye97]. These examples will be used later to help describing our model and comparing it with other related work.



3.1 A Medical Data Analysis Example

Consider a medical data analysis system illustrated in Figure 8. The purpose of the medical system is to permit statistical analysis on a large number of patient records. While patients would like to keep their own patient information private, they may allow their own patient history to be accessed by statisticians and researchers, provided that they perform statistical research without exposing personal information to public.

In Figure 8, there are four important principals: patient P_1 and P_2 , researcher R , and statistician S . There are also two kinds of tools: patient record extractor (PRE) and analysis package (AP), whose purpose are to make a brief summary of and to produce a statistical report based on patient records, respectively.

The label structure has the form: $\{P_1: L_1, P_2: L_2, P_3: L_3 \dots\}$, meaning the subject P_i classifies the labeled object as level L_i according to his/her own judgment. Since the patient record is labeled $\{P_1: H, P_2: H, R: M, S: L\}$ in this case, it is considered private

from both P_1 and P_2 perspective since both label the object as H (High). Likewise, the analysis result is labeled $\{P_1: L, P_2: L, R: L, S: L\}$. In such a setting, information can not flow from patient records to analysis report.

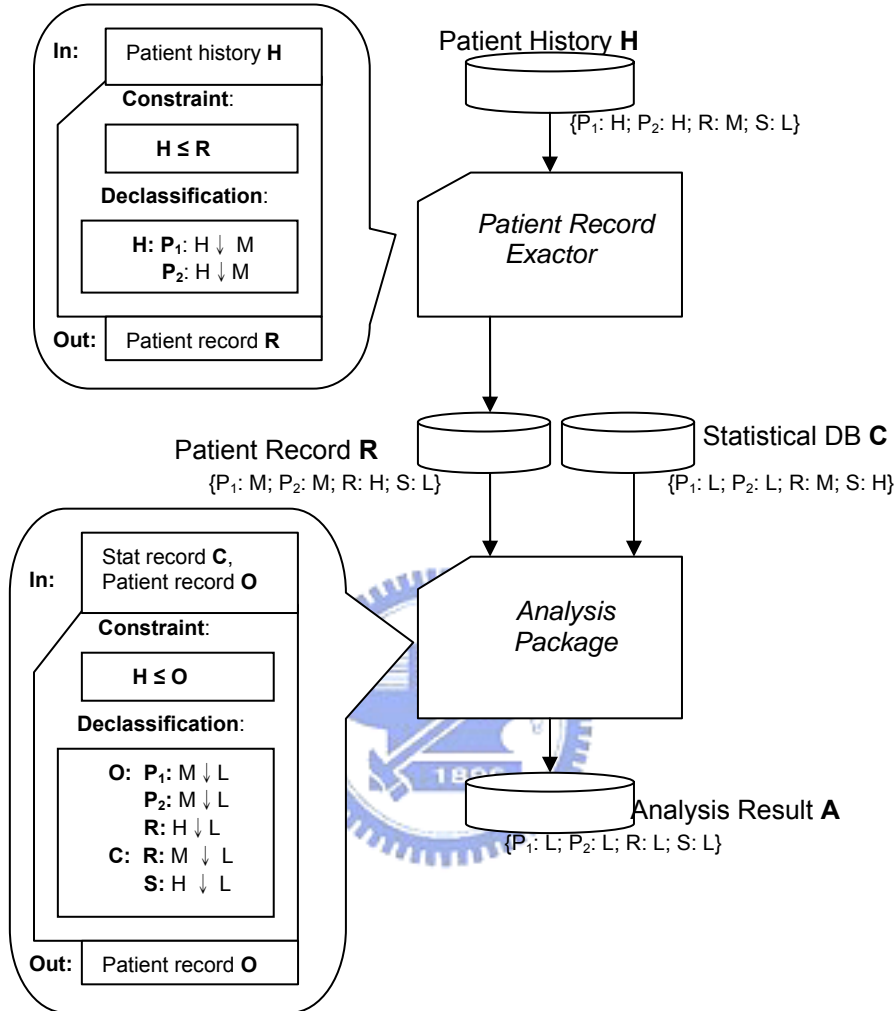


Figure 8. A medical data analysis scenario where statistical analysis is conducted on patient records.

In this model, a label of a tool contains a parameter set, a constraint set for parameters, and a declassification specification. For example, the utility tool PRE exacts information from patient histories and summarizes them as patient records. PRE is labeled for each of its parameters: the input patient history is labeled H and the output summary is labeled R. The constraint set states that R should be more restrictive than H, whereas the declassification specification declares that the input with label R can be dropped to M temporarily, for both P_1 and P_2 , provided that the actual output has label the same or

more restrictive than M. Similarly, the results of the statistical study are made public for all users with patient records kept private, with the understanding that AP would not leak private information through its output. In this way statistic record and patient record can flow to analysis result when AP is invoked. Note that users declassify tools individually, and changing the declassification specification of one tool does not affect the specifications of other tools. However, since a tool may invoke other tools during its own computation, automated analysis of the chain of tool invocations is needed to determine whether information flow is violated in the presence of declassification.

The example above shows that it is possible to sharing private information with other users without giving all privileges to them.

3.2 A Bank Example

Consider another bank example, as shown in Figure 9, where the bank serves many customers and both the bank and the customers would like to keep their assets private. The bank would like to keep its own total asset safe from all customers and the customers would like to keep their own asset safe from other customers.

This example contains three subjects: bank B and customers C_1 and C_2 . One requirement of this example is that depositing money into personal account and updating the total bank deposit are done without leaking individual account information to unexpected customers and non-customers. There are also two tools in this example: the asset depositor (AD) and total asset updater (TAU), which take account information and deposit money into individual asset, and updates the total bank asset, respectively. A customer can make deposit request through these tools.

The object `DepositRequest` of C_2 is labeled $\{B: L, C_1: L, C_2: H\}$ because that the deposit is requested by principal C_2 . C_2 can always use the tool AD to deposit money into his account. The deposit request should contain information about his account, or the invoking of depositor would fail.

Since the functionality of AD is to deposit money into a personal account, the constraint of AD is that the resulting asset should be more restrictive than the deposit. Now asset is labeled as $\{B: M; C_1: L; C_2: H\}$ which is more restrictive than the deposit, which is labeled $\{B: L; C_1: L; C_2: H\}$, so declassification is not needed here.

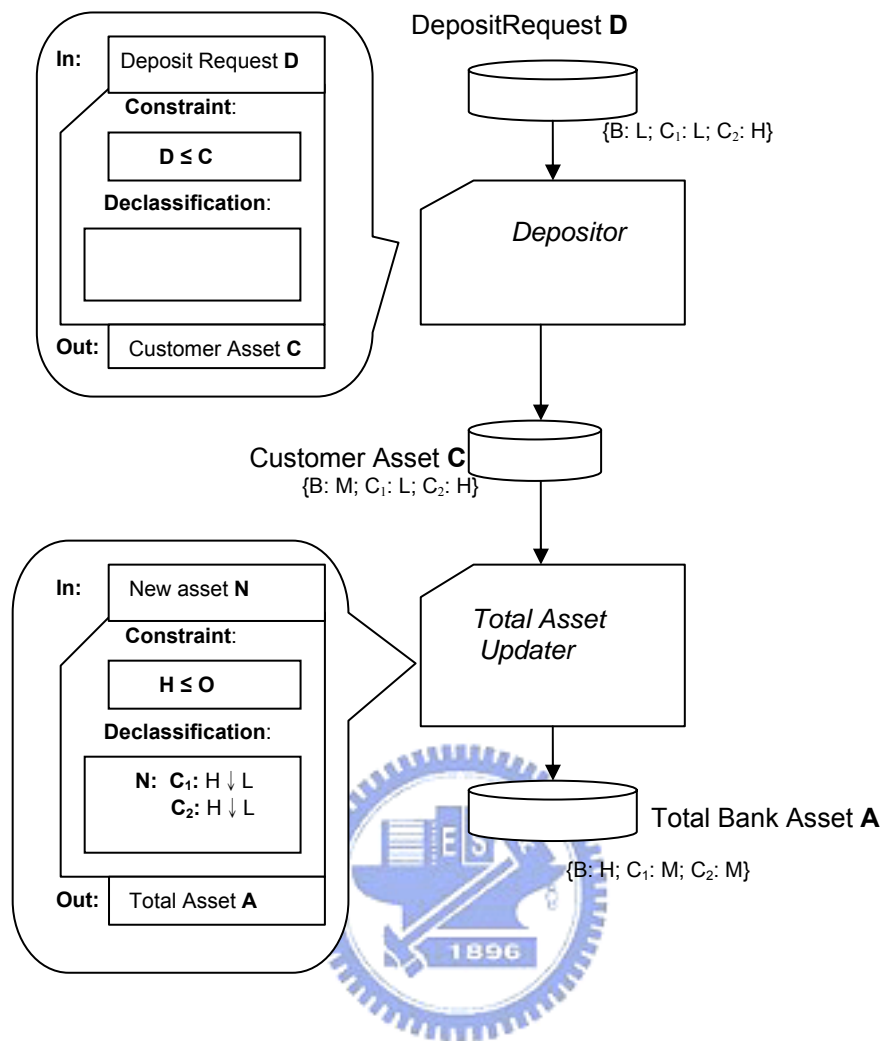


Figure 9. The bank example where each customer keeps asset information private from other customers and the bank wants to keep its total asset private.

After completing deposit transactions for individual customers, the bank needed to update its total asset, which is done by TAU. However, the constraint set of TAU states that the label of the total bank asset must be more restrictive than the label of individual asset, which goes against the object labeling where the total bank asset is labeled $\{B: H; C_1: M; C_2: M\}$ and is less restrictive than the label of asset C_2 with label $\{B: M; C_1: L; C_2: H\}$.

The conflict can be solved by proper declassification, by the customers, such that the customer asset can be dropped to level L if it has level H originally. Thus the total asset of bank can be updated when customers can control propagation of their own account and asset information.

This scenario shows a bank example making processing normally and also keeping each principal's information private without any rigid constraints. Customers can keep their own personal account information and asset private from other customers and non-customers.



Chapter 4 A Rule-Based Decentralized Information Flow Approach

In this section we define a decentralized model fulfilling the objective discussed in the previous section, by allowing individual principals to specify their own policies based on user-defined lattice structures. This model handles declassification in a selective way, through tool declassification, rather than by giving all access rights to other principals. As a result, policy management becomes easier to handle with our model.

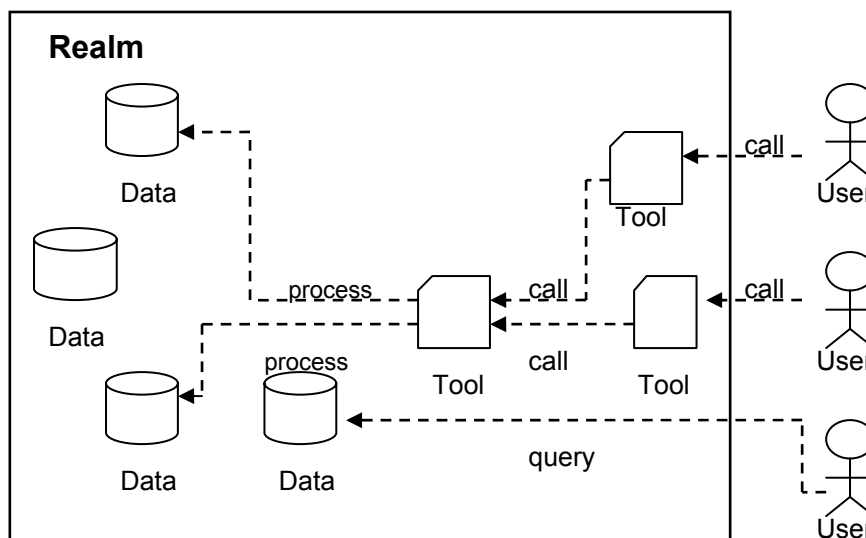


Figure 10. Overview of our information sharing model, that consists of a set of principals, data, and tools.

4.1 Overview

Figure 10 depicts our overall secure information sharing model, which is basically a *realm* of secure environment that contains a set of (data) *objects*, a set of *tools*, and a set of *principals* representing users. Within the realm, objects are created and manipulated only through tools, whereas tools are created by principals only. Specifically, principals do not access objects directly but through tools instead. Technically, to access tools, each user needs to first establish a communication channel with the realm, during the

process the channel is bound with the corresponding principal and assumes the identity and associated privileges of that principal.

Each object inside the realm is attached with a label that is further divided into multiple sub-labels each corresponding to a principal. In other words, each principal assigns a sub-label to each object representing the security level of the object from the principal's perspective, and the total sub-labels to the object constitute the actual label.

Tools are considered an integral part in our model. Informally, each tool is associated with certain documentation indicating what the tool do and the information flow it induces when invoked. A tool can be invoked only when its execution will not violate the information flow requirement, i.e. causing information flowing from high-level objects to low-level ones.

In our model, data are imported into the realm through tools. The label of the imported data is decided according to the setting of declassification. For example, a doctor can specify declassification on the tool which is used for importing new data by statisticians about the output values imported from the tool can rise to level M. This approach is similar to importing data through channels in decentralized label model.

4.2 Object and Principal Labeling

As mentioned, security labels represent the security levels of the objects they denote from the perspective of all participating principals, respectively. Different from other models, we view principals as a special kind of objects which information may flow from and into. Whether a principal can read or modify the content of an object is subject to the same information flow constraint. Furthermore, a principal can assign security labels to other principals in exactly the same way he/she labels objects.

Our model excludes the concept of ownership – objects are considered shared and administrated by all principals “collectively.” As illustrate in the Figure 11, the label of a diagnosis record contains sub-labels P_1 , P_2 , and P_3 which represents security level of patient, statistician, and researcher, respectively.

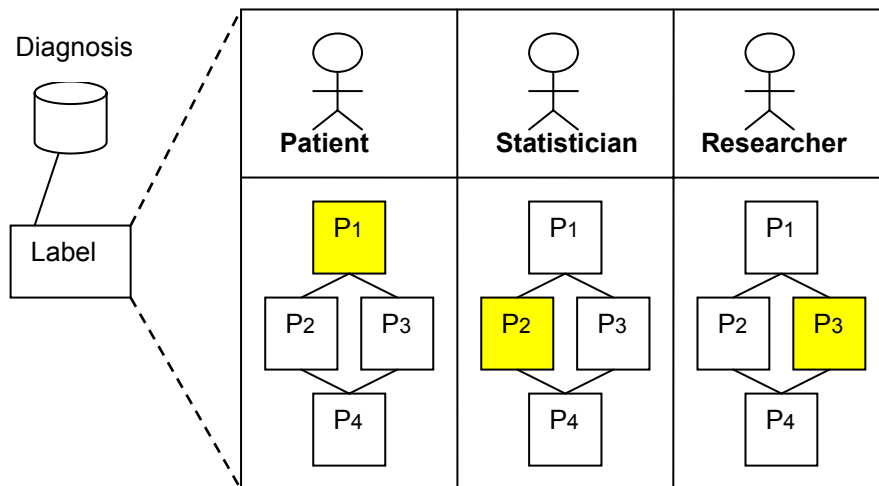


Figure 11. An object label contains sub-labels each indicating the security level of the object from each principal's perspective.

Figure 12 illustrate the general lattice structure of object labels. It is assumed that the sub-labels coming from the same principal are drawn from a lattice defined by that principal so that the can-flows relations among labels can be defined, as shown in Figure 13. The rule from L to L' must be restriction which implies each sub-label of L should be more restrictive than L' . This rule for information flow is more precise because the label is formed as Cartesian product of sub-labels of all principals.

Definition of the label structure:

$$L = \{ p_1:l_1; p_2:l_2; p_3:l_3; \dots \}$$

where p_i represents a principal and $l_i \in L_i$ a sub-label drawn from the lattice L_i associated with p_i .

Figure 12. Definition of the label structure.

Definition of the can-flows relation from L to L' :

$$L = (p_1:l_1; p_2:l_2; \dots) \leq L' = (p_1:l_1'; p_2:l_2'; \dots) \text{ iff } l_i \leq l_i' \forall i$$

Figure 13. Definition of the can-flows relation

When the content of an object is derived from the contents from other objects, such derivation should respect the information flow constraint. For example, consider a computation $x = y + z$ where x would contain information about y and z , the label of x should be more restrictive than y and z . More generally, if an object labeled L contains information derived from the objects labeled L_1 and L_2 , respectively, L should be more restrictive than both L_1 and L_2 . The least restrictive label that is more restrictive than L_1 and L_2 , denoted $L_1 \cap L_2$, is defined as follows:

Definition of the join operator $L \cap L'$:

$$L \cap L' = (p_1:l_1 \cap l_1'; p_2:l_2 \cap l_2'; \dots)$$

$$\text{where } L = (p_1:l_1; p_2:l_2; \dots), L' = (p_1:l_1'; p_2:l_2'; \dots)$$

and $l_i \cap l_i'$ the join of l_i and l_i' defined in L_i

Figure 14. Definition of the join operator among labels

It is easy to see that the labels in our model form a lattice, since the can-flows relation is a partial order relation, with the least restrictive label denoted as \perp , and the most restrictive label denote as \top well defined.



4.3 A Rule-Based Labeling Approach

In our model, instead of labeling objects individually, a principal associates labels with objects through user-defined predicates. A predicate is a rule-based assignment of a sub-label to a subset of objects within the realm. Specifically, the subset is defined through a set of user-defined rules. Using the example in Figure 11 for illustration, the patient may define a predicate stating that (from the patient's perspective) "all diagnosis information is P_1 ," then all diagnosis information of the patient in the realm would be attached P_1 . The situation is similar for the statistician and the researcher, with diagnosis attached P_2 and P_3 , respectively. In this thesis, we leave concrete definition of such rules open, although we have implemented some rule-based labeling mechanisms using Prolog. However, we do require that any such rule-based predicate mechanism should partition the objects statically, such that once an object is attached a label, such assignment never change. A simpler rule of thumb is that object labeling should base solely on

object identities without relying on object contents or other external (dynamic) environmental conditions. For example, assuming all medical data are stored in a file system, a predicate by a doctor may prescribe that all records in the directory storing diagnoses are all labeled private. In this example, the directory is part of the object identities for all the diagnoses, and the predicate even asserts that “future” diagnosis records to be placed in the directory will have the same security level from the doctor’s perspective.

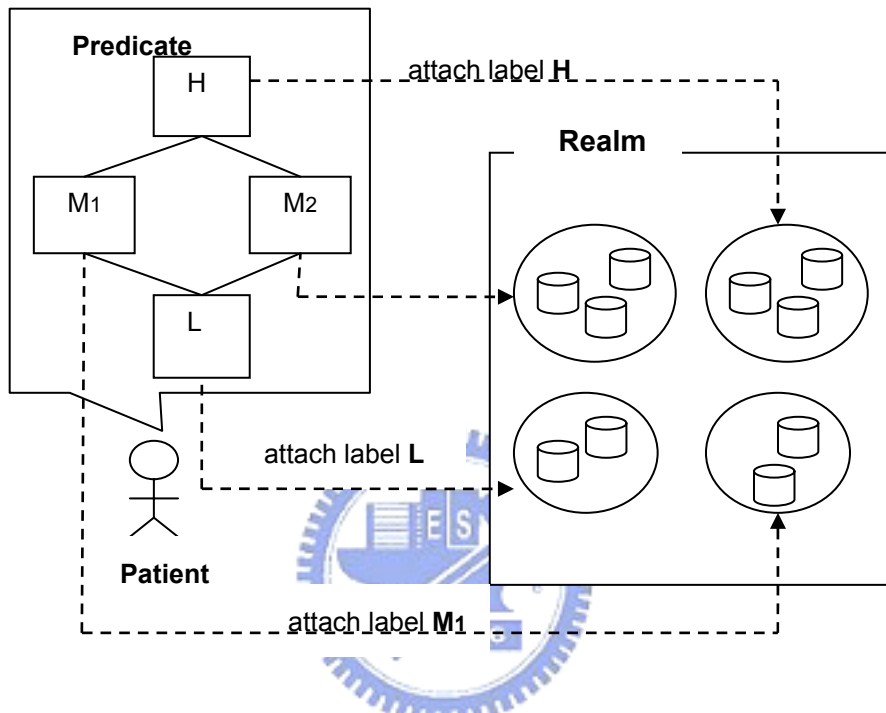


Figure 15. Our model allows a principal to attach individual labels on data in the realm.

4.4 Tool Sharing and Declassification

A tool executes by processing data from its input parameters (or channels) and generates output into its output parameters. Therefore, a tool should contain explicit *document* about the information flow it may cause among its input/output parameters. For “system” tools, possibly written in system programming languages, the system administrator is responsible of ensuring that these tools do not cause information flow exceeding what they claim in their documents. Therefore, system tools are outside the control of principals and used as is. However, principals can still create tools using the programming languages allowed by the system, as long as the system can analyze and gen-

erate correct documents for these tools automatically. We will describe a simple programming language for tool definition in the next chapter.

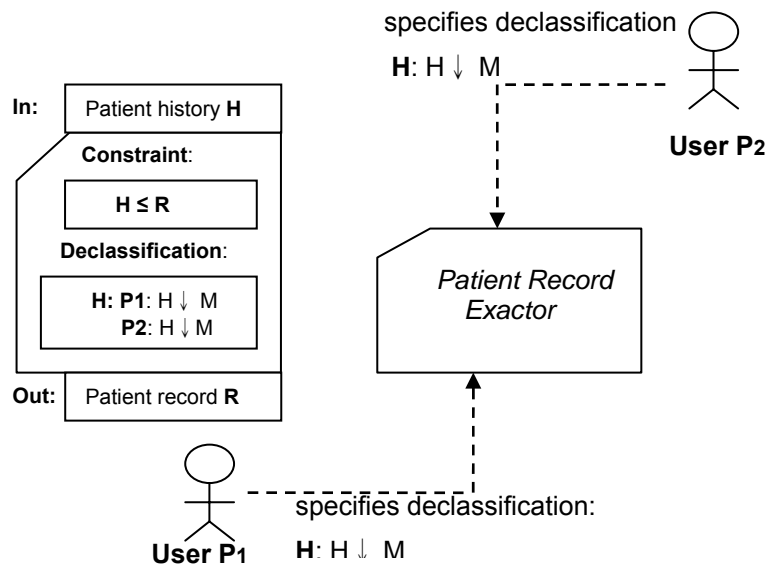


Figure 16. Each user can specify declassification on each tool.

An example of information flow documents is shown in Figure 16, where the constraint set of the tool PRE indicates that information would flow from patient history to patient record. That is, the label of patient record should be more restrictive than patient history; otherwise, the invocation of the tool would be denied by the system. The signature of tools helps user recognize the constraints for what flow would happen after using the tool and also help users for specify declassification.

The purpose of declassification is to relax the constraints of tools so as to enable more useful data analysis. Declassification refers to the permission of information flow from higher-level objects to lower-level ones. In earlier systems for controlling information flow, declassification relies on the trust relation outside the model. In our model, on the other hand, declassification comes with tools. For example, in Figure 16, a patient may specify declassification such that the output data patient record can raise level from M to H temporally. Originally, information flow from patient history to patient record is denied because that input parameter corresponding to patient record is lower than M in the patient's part. After specifying declassification by patient, the constraint of the tool

is relaxed such that information can flow from patient history which labeled level L patient record which labeled level M in processing of the tool patient record extractor.

In summary, our model represents a flexible approach to sharing and protection of information in a decentralized manner. We use consistent labeling approach base on lattice to specify policies on data and other user. Unlike existing models focus on who can share and declassify information, our model focuses on how to share and declassify. Without giving all access rights to another parties, our model deal declassification in a safer way. Moreover, the use of predicates also reduces the complexity of managing policies for large volume of data.



Chapter 5 Type Systems Approach to Information Flow

In this chapter we present a program certification mechanism by defining a programming language for tools and developing a verifier for the language. There are three main advantages of program certification mechanisms over run-time enforcement mechanisms [Den76]. First, to prevent purposely security violations, the program execution is guaranteed to be secure before it executes. Second, a certification mechanism would not blemish the speed of program execution since all security checks are performed before the program execution. Third, to be comprehended and corrected conveniently, the certification mechanism can be specified in terms of higher level languages rather than low level hardware instruction. We present our program certification framework via type theoretical approach following Volpano's method. In particular, we define the syntax and operational semantics of a simple programming language and augment it with declassification specification..

5.1 The Volpano Approach

Volpano [Vop96, Vop97] adopts a type system approach to information flow analysis. A type system for a procedural language guarantees that well-typed programs do not cause undefined value conversions during execution. In the context of information flows, the type system approach is adapted so that well-typed programs do not cause information flow from high-level objects to low-level ones, which are often termed non-interference property. In short, a system has the noninterference property if no matter how high-level data object change during computation, low-level data objects remain the same. Using the following procedure contains two parameters as an example,

```
proc p( in x: high , out y: low )
```

where x and y are variables with high and low security levels, respectively. Now two calls p(a: high, b: low) and p(c: high, b: low) end with final values a, b, and c. If the procedure p is noninterference, then a and c may change and the final value of b would remain the same in both cases no matter how a and c change.

The syntax definition of the Volpano model is given below:

(Phrase) $p ::= e \mid c$
 (Expr) $e ::= x \mid n \mid l \mid e + e' \mid e - e' \mid e = e' \mid e < e' \mid$
 proc (**in** x_1 , **inout** x_2 , **out** x_3) c
 (comm) $c ::= e := e' \mid c; c' \mid e(e_1, e_2, e_3) \mid$
 if e **then** c **else** c' **when** e **do** $c \mid$ **letvar** $x := e$ **in** $c \mid$
 letproc x (**in** x_1 , **inout** x_2 , **out** x_3) c **in** c'

where meta-variable x ranges over identifiers, n ranges over integer literals and l ranges over external storage locations which can be used for input and output in the language. The initial values of external locations represent inputs to a program and final values of external locations represent the output of the program.

Variables in a program should be labeled to denote its security level. For example, each variable x , y has a security class denoted as \underline{x} , \underline{y} , respectively. Information flow from x to y is permitted if and only if $x \leq y$. The type definition of core language in the Volpano model is shown below,

$$\begin{aligned} \tau &::= s \\ \pi &::= \tau \mid \tau \text{ proc } (\tau_1, \tau_2 \text{ var}, \tau_3 \text{ acc}) \mid \tau \text{ cmd} \\ \rho &::= \pi \mid \tau \text{ var} \mid \tau \text{ acc} \end{aligned}$$

where meta-variable s ranges over a set of security levels which is partial ordered by \leq .

The typing of variables, i.e. assignment of labels to variables, is done through type inference. Consider the assignment $x := y$. If the identifier typing γ give x low and give y high, that is, $x \leq y$, then the assignment is rejected by the typing rule since the content in variable y would affect the final value of x . Thus a typing rule is introduced to cover assignment:

$$\frac{\begin{array}{l} \lambda; \gamma \vdash e : \tau \text{ var} \\ \lambda; \gamma \vdash e' : \tau \end{array}}{\lambda; \gamma \vdash e := e' : \tau \text{ cmd}}$$

The rule says that if the identifier typing γ give e τ var and give e' τ var then the command $e := e'$ is typed as τ cmd. As another example, suppose that we try to copy x to y indirectly as follows:

```

while (  $x > 0$  ) do
   $y = y + 1;$ 
   $x = x - 1$ 
end

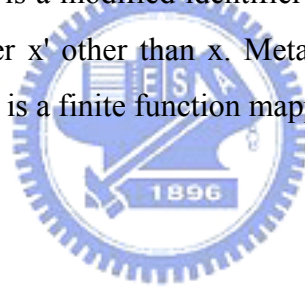
```

The final value of y is affected by x . This is so-called implicitly information flow from x to y . Thus it is needed to insist typing rule for the while statement such that the guard and the body should be typed at the same security level:

$$\frac{\begin{array}{l} \lambda; \gamma \vdash e : \tau \\ \lambda; \gamma \vdash c : \tau \text{ cmd} \end{array}}{\lambda; \gamma \vdash \mathbf{while } e \mathbf{ then } c : \tau \text{ cmd}}$$

It is possible check a program is well type by security type inference. Security type inference attempted to use type variables to presents unknown types and collect constraints which are in the form of type inequalities such that the type variables must satisfy for the program to be well typed. The typing rules of the Volpano model are given in Figure 17.

In the typing rule $\lambda; \gamma \vdash p : \rho$, meta-variable γ ranges over identifier typing. An identifier typing is a finite function mapping identifiers to types of the form τ ; $\gamma(x)$ is the type assigned to x by γ , and $\gamma[x: \rho]$ is a modified identifier typing assigning type ρ to x and assigning $\gamma(x')$ to any identifier x' other than x . Meta-variable λ ranges over location typing, where a location typing is a finite function mapping from locations to τ types.



(letvar)	$\lambda; \gamma \vdash e : \tau$ $\lambda; \gamma [x: \tau \text{ var}] \vdash c : \tau' \text{ cmd}$	$\lambda; \gamma \vdash \mathbf{letvar} \ x := e \ \mathbf{in} \ c : \tau' \text{ cmd}$
(if)	$\lambda; \gamma \vdash e : \tau$ $\lambda; \gamma \vdash c : \tau \text{ cmd}$ $\lambda; \gamma \vdash c' : \tau \text{ cmd}$	$\lambda; \gamma \vdash \mathbf{if} \ e \ \mathbf{then} \ c \ \mathbf{else} \ c' : \tau \text{ cmd}$
(while)	$\lambda; \gamma \vdash e : \tau$ $\lambda; \gamma \vdash c : \tau \text{ cmd}$	$\lambda; \gamma \vdash \mathbf{while} \ e \ \mathbf{then} \ c : \tau \text{ cmd}$
(procedure)	$\lambda; \gamma [x_1 : \tau_1, x_2 : \tau_2 \text{ var}, x_3 : \tau_3 \text{ acc}] \vdash c : \tau \text{ cmd}$	$\lambda; \gamma \vdash \mathbf{proc} \ (\mathbf{in} \ x_1, \mathbf{inout} \ x_2, \mathbf{out} \ x_3) \ c :$ $\tau \text{ proc}(\tau_1, \tau_2 \text{ var}, \tau_3 \text{ acc})$
(apply)	$\lambda; \gamma \vdash e : \tau \mathbf{proc} \ (\tau_1, \tau_2 \text{ var}, \tau_3 \text{ acc})$ $\lambda; \gamma \vdash e_1 : \tau_1 ; \lambda; \gamma \vdash e_2 : \tau_2 \text{ var} ; \lambda; \gamma \vdash e_3 : \tau_3 \text{ acc}$	$\lambda; \gamma \vdash e \ (e_1, e_2, e_3) : \tau \text{ cmd}$
(letproc)	$\lambda; \gamma \vdash \mathbf{proc} \ (\mathbf{in} \ x_1, \mathbf{inout} \ x_2, \mathbf{out} \ x_3) \ c : \pi$ $\lambda; \gamma \vdash [\mathbf{proc} \ (\mathbf{in} \ x_1, \mathbf{inout} \ x_2, \mathbf{out} \ x_3) \ c/x] \ c' : \tau \text{ cmd}$	$\lambda; \gamma \vdash \mathbf{letproc} \ x(\mathbf{in} \ x_1, \mathbf{inout} \ x_2, \mathbf{out} \ x_3)c \ \mathbf{in} \ c' : \tau \text{ cmd}$

Figure 17. Typing rules for the Volpano model

The evaluation rules for the Volpano model are shown in the Figure 18, in which a closed phrase is evaluated relative to a memory μ . An evaluation is a finite function mapping from location to integers. The semantics uses $\mu \vdash e \Rightarrow n$ for the expression evaluation and $\mu \vdash c \Rightarrow \mu'$ for the command evaluation. The content of a location $l \in \text{dom}(\mu)$ is the integer $\mu(l)$, and $\mu[l := n]$ denotes that memory assigns n to location l and assigns $\mu(l')$ to location l' other than l . They write $[e'/x]$ to denote the capture-avoiding substitution of e' for all free occurrences of x in c .

(val)	$\mu \vdash n \Rightarrow n'$
(contents)	$\mu \vdash l \Rightarrow \mu(l) \quad l \in \text{dom}(\mu)$
(add)	$\frac{\begin{array}{l} \mu \vdash e \Rightarrow n \\ \mu \vdash e' \Rightarrow n' \end{array}}{\mu \vdash e + e' \Rightarrow n + n'}$
(sequence)	$\frac{\begin{array}{l} \mu \vdash c \Rightarrow \mu' \\ \mu' \vdash c' \Rightarrow \mu'' \end{array}}{\mu \vdash c; c' \Rightarrow \mu''}$
(branch)	$\frac{\begin{array}{l} \mu \vdash e \Rightarrow 1 \\ \mu \vdash c \Rightarrow \mu' \end{array}}{\mu \vdash \mathbf{if} \ e \ \mathbf{then} \ c \ \mathbf{else} \ c' \Rightarrow \mu'}$ $\frac{\begin{array}{l} \mu \vdash e \Rightarrow 0 \\ \mu \vdash c' \Rightarrow \mu' \end{array}}{\mu \vdash \mathbf{if} \ e \ \mathbf{then} \ c \ \mathbf{else} \ c' \Rightarrow \mu'}$
(update)	$\frac{\begin{array}{l} \mu \vdash e \Rightarrow n \\ l \in \text{dom}(\mu) \end{array}}{\mu \vdash l := e \Rightarrow \mu'[l := n]}$
(update)	$\frac{\begin{array}{l} \mu \vdash e \Rightarrow n \\ l \in \text{dom}(\mu) \end{array}}{\mu \vdash l := e \Rightarrow \mu'[l := n]}$
(loop)	$\frac{\mu \vdash e \Rightarrow 0}{\begin{array}{l} \mu \vdash \mathbf{while} \ e \ \mathbf{do} \ c \Rightarrow \mu \\ \mu \vdash e \Rightarrow 1 \\ \mu \vdash c \Rightarrow \mu' \\ \mu' \vdash \mathbf{while} \ e \ \mathbf{do} \ c \Rightarrow \mu'' \end{array}}$ $\mu \vdash \mathbf{while} \ e \ \mathbf{do} \ c \Rightarrow \mu''$

Figure 18. Evaluation rules for the Volpano model

5.2 Type System of Our Model

In this section we follow Volpano’s approach and describe our model by defining a type system. The syntax definition of our language is given below:

(phrase)	$\rho ::= e \mid c$
(expression)	$e ::= x \mid l \mid n$
(commands)	$c ::= e := e' \mid c; c' \mid e(e_1, e_2) \mid$ if e then c else $c' \mid$ when e do $c \mid$ letvar $x = e$ in $c \mid$ lettool $x(\mathbf{in} \ x_1, \mathbf{inout} \ x_2) c$ in c'

Figure 19. Language syntax definition

In the definition, meta-variable x ranges over identifiers, n ranges over integer literals, and l ranges over external locations (such as files). Like in the Volpano model, the initial contents of the external locations represent inputs and final values of external resources represent the output. For simplicity, we assume that each tool has two parameters: the parameters of type **in** cannot be assigned values to while the parameters of type **inout** can. Therefore, **inout** parameters accept only locations but not literal values.

The typing rules of our model are shown in Figure 20. In a typing rule $\theta \vdash e : \theta'$, meta-variable θ ranges over contexts. A context contains information about the total security level of the “environment” under which e is evaluated, and contains a set of bound variables in actuality. That is, intuitively, all expressions or commands under a context can see the contents of all the variables in that context. In the typing rule $\phi; \theta \vdash c : \phi'$, meta-variable ϕ ranges over flow constraints. A flow constraint is a finite set of dependencies among variables, which is defined (informally) below.

Definition 5.2.1 (Flow Declaration): for a flow constraint ϕ and a variable x , $\phi[x]$ is a flow constraint extended with the declaration of x (Figure 21b). Intuitively, declaration introduces a fresh variable x shadowing those in ϕ , so that it captures the occurrences of x introduced later on when additional flow constraints are included.

(assign)	$\theta \vdash e : \theta'$
	$\phi; \theta \vdash x := e : \phi[x \leftarrow \theta']$
(compose)	$\phi; \theta \vdash c : \phi' \quad \phi'; \theta \vdash c' : \phi''$
	$\phi; \theta \vdash c; c' : \phi''$
(if)	$\theta \vdash e : \theta'$ $\phi; \theta' \vdash c_1 : \phi'$ $\phi'; \theta' \vdash c_2 : \phi''$
	$\phi; \theta \vdash \mathbf{if}(e, c_1, c_2) : \phi''$
(while)	$\theta \vdash e : \theta'$ $\phi; \theta' \vdash c : \phi''$
	$\phi; \theta \vdash \mathbf{while}(e, c) : \phi''$
(letvar)	$\theta \vdash e : \theta'$ $\phi[x][x \leftarrow \theta']; \theta' \vdash c : \phi''$
	$\phi; \theta \vdash \mathbf{let } x = e \mathbf{ in } c : \phi'' \langle x \rangle$
(call)	$\theta \vdash e : \theta' \quad \theta \vdash l : \theta''$ $\{ \} [x_1][x_2]; \{ \} \vdash c : \phi'$ $\phi'' = \phi[x_1][x_2][\phi'[\theta'/x_1][\theta''/x_2]] \langle x_2 \rangle \langle x_1 \rangle$
	$\phi; \theta \vdash (\mathbf{tool}(\mathbf{in } x_1, \mathbf{inout } x_2) c)(e, l) : \phi''$

Figure 20. Typing rules for our model

Definition 5.2.2 (Flow Constraints): for a flow constraint ϕ , a context θ , and a variable x , $\phi[x \leftarrow \theta]$ is a flow constraint with additional flow dependencies from θ to x , that is, from all variables in θ to x (Figure 21c)

Definition 5.2.3 (Flow Un-declaration): for a flow constraint ϕ and a variable x declared in ϕ , $\phi \langle x \rangle$ is a flow constraint by removing the declaration of x and “redirect” all flow dependencies into x in ϕ to all the other variables in ϕ that depends on x (Figure 21d).

Definition 5.2.4 (Flow Instantiation): for a flow constraint ϕ , a context θ , and a variable x declared in ϕ , $\phi[\theta/x]$ is a flow constraint in which each occurrence of x are replaced with all variables from θ in ϕ (and the dependencies “inherited”).

Definition 5.2.5 (Flow Embedding): for flow constraints ϕ and ϕ' , $\phi[\phi']$ is a flow constraint such that, $\phi[\phi'] = \phi [x_1 \leftarrow \theta_1] [x_2 \leftarrow \theta_2] \dots$ for all $x_i \leftarrow \theta_i$ in ϕ' . Intuitively, flow embedding joins two flow constraints by joining all their variable dependencies together.

Figure 21 illustrate diagrammatically the typing rule (letvar) involving flow declaration and undeclaration. The typing rule (letvar) first declares flow variable x by $\phi [x]$, then analyzes flow from context θ to x written as $\phi [x] [x \leftarrow \theta]$ and undeclares variable x by $\phi [x] [x \leftarrow \theta] \langle x \rangle$ finally.

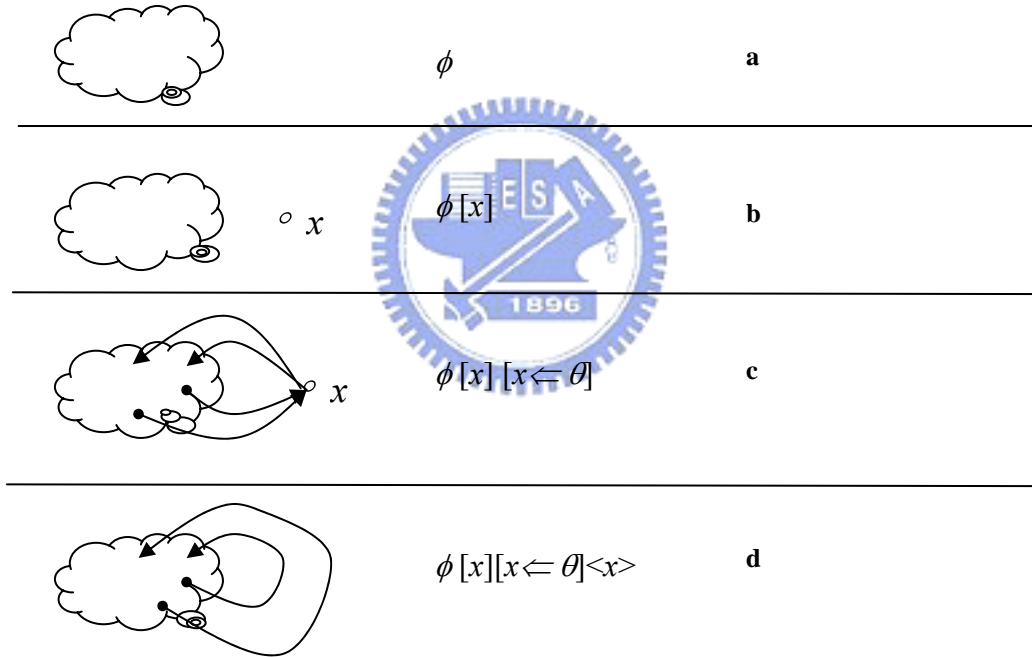


Figure 21. Flow declaration and un-declaration for typing rule (**letvar**)

The evaluation rules are shown in the Figure 22. A closed phrase is evaluated relative to a memory μ . An evaluation is a finite function mapping from location to integers. Our semantics uses $\mu \vdash e \rightarrow n$ for the expression evaluation and $\mu \vdash c \rightarrow \mu'$ for the command evaluation. The content of a location $l \in dom(\mu)$ is the integer $\mu(l)$, and $\mu[l := n]$ denotes that memory assigns n to location l and assigns $\mu(l')$ to location l' other than l .

We write $[e'/x]$ to denote the capture-avoiding substitution of e' for all free occurrences of x in c .

(val)	$v; \mu \vdash n \Rightarrow n'$
(contents)	$v; \mu \vdash l \Rightarrow \mu(l) \quad l \in \text{dom}(\mu)$
(variable)	$v; \mu \vdash x \Rightarrow v(x) \quad x \in \text{dom}(v)$
(add)	$v; \mu \vdash e \Rightarrow n$ $v; \mu \vdash e \Rightarrow n'$
<hr/>	
(sequence)	$v; \mu \vdash e + e' \Rightarrow n + n'$ $v; \mu \vdash c \Rightarrow v'; \mu'$ $v'; \mu' \vdash c' \Rightarrow v''; \mu''$
<hr/>	
	$v; \mu \vdash c; c' \Rightarrow v''; \mu''$
(branch)	$v; \mu \vdash e \Rightarrow 1$ $v; \mu \vdash c \Rightarrow v'; \mu'$
<hr/>	
	$v; \mu \vdash \mathbf{if\ } e \mathbf{ then\ } c \mathbf{ else\ } c' \Rightarrow v'; \mu'$
	$v; \mu \vdash e \Rightarrow 0$ $v; \mu \vdash c' \Rightarrow v'; \mu'$
<hr/>	
	$v; \mu \vdash \mathbf{if\ } e \mathbf{ then\ } c \mathbf{ else\ } c' \Rightarrow v'; \mu'$
(update)	$v; \mu \vdash e \Rightarrow n \quad l \in \text{dom}(\mu)$
<hr/>	
	$v; \mu \vdash l := e \Rightarrow v; \mu'[l := n]$
(loop)	$v; \mu \vdash e \Rightarrow 0$
<hr/>	
	$v; \mu \vdash \mathbf{while\ } e \mathbf{ do\ } c \Rightarrow v; \mu$
	$v; \mu \vdash e \Rightarrow 1$ $v; \mu \vdash c \Rightarrow v'; \mu'$ $v'; \mu' \vdash \mathbf{while\ } e \mathbf{ do\ } c \Rightarrow v''; \mu''$
<hr/>	
	$v; \mu \vdash \mathbf{while\ } e \mathbf{ do\ } c \Rightarrow v''; \mu''$

$$\begin{array}{l}
\text{(call)} \quad \frac{v; \mu \vdash e \Rightarrow n \quad v; \mu \vdash [n, l/x_1, x_2]c \Rightarrow v'; \mu'}{v; \mu \vdash (\mathbf{proc}(\mathbf{in} x_1, \mathbf{inout} x_2) c)(e, l) \Rightarrow v'; \mu'} \\
\text{(bindvar)} \quad \frac{v; \mu \vdash e \Rightarrow n \quad y \text{ is a fresh variable not used during evaluation} \quad v[y=n]; \mu \vdash [y/x]c \Rightarrow v'; \mu'}{v; \mu \vdash \mathbf{letvar} x = e \mathbf{in} c \Rightarrow v'; \mu'} \\
\text{(bindproc)} \quad \frac{v; \mu \vdash [\mathbf{proc}(\mathbf{in} x_1, \mathbf{inout} x_2) c / x] c' \Rightarrow v'; \mu'}{v; \mu \vdash \mathbf{letproc} x(\mathbf{in} x_1, \mathbf{inout} x_2) c \mathbf{in} c' \Rightarrow v'; \mu'}
\end{array}$$

Figure 22. Evaluation rules in our model

In this thesis we only give informal treatment of the type system. We have implemented an inference algorithm conforming to the typing rules described above using Prolog and tested our implementation on many scenarios, including those described in the ML model. Further analysis for the correctness of the algorithm as well as the type system as a whole is necessary, and is left as future work.

Chapter 6 Comparisons with the ML Model

In this chapter, we compare our model with ML by discussing several examples described in [Mye97, Mye98] and showing how to use our model to achieve similar access control policies.

The Password Checking Example

The example password checker in [Mye97] shows that a procedure checks whether the input username and password are consistent by comparison to db, as shown in Figure 23.

```
chek_password (db:array[ ], user:string, pwd:string)
return(return: boolean {  $\cup$ user  $\cup$ pwd  $\cup$ db })

  i : int=0
  match : bool = false
  while i<db.length() do
    if db[i].names = user &
      db[i].passwords = pwd then
      match = true
    end
    i = i +1
  End
  return = false
  If_acts_for(check_password,checker) then
    return = declassify( match, {client:chkr})
  end

End check_password
```

Figure 23. Checking password example which is a procedure to check whether the username and password is correct.

Two principals are mentioned in this example: checker who acts a password checker, and client who calls the procedure. The input parameters of this procedure include a **password database** containing records of user names and corresponding passwords, **username** and **password** inputted by client. This procedure would return a result indicating whether password and username are matched against the password database.

To prevent information leakage, this procedure is checked statically for generating constraints for invoking this procedure. Figure 24 shows the constraints for the procedure. Constant variables are all labeled \perp which represents the least restrictive label. $L_1 \subseteq L_2$ represents relabeling from L_1 to L_2 , which is legal when L_1 contains more readers and few owners than L_2 . Note that L_d represents the label of the declassification result, $\underline{\text{match}} \subseteq L_d \cup \{\text{chkr}: \emptyset\}$ indicating that the declassification is legal if not effecting reader sets of other owners.

	constraints	
i	\perp	$\subseteq i$
match = F	\perp	$\subseteq \text{match}$
while	$\perp \cup \text{db}$	$= L_1$
if	$i \cup \text{user} \cup \text{pwd} \cup \text{db} \cup \{\text{chkr}: \text{chkr}\} \cup L_1$	$= L_2$
match=T	$L_1 \cup L_2$	$\subseteq \text{match}$
i = i+1	$i \cup L_1$	$\subseteq i$
return = F	\perp	$\subseteq \text{user} \cup \text{pwd} \cup \text{db}$
Declassify	match	$\subseteq L_d \cup \{\text{chkr}: \emptyset\}$
return	L_d	$\subseteq \text{user} \cup \text{pwd} \cup \text{db}$

Figure 24. Constraints of the password checking example presented by decentralized label model.

Figure 25 shows the solution of the constraints of the password example. Input parameters should satisfy the solution of the constraints or the invoking of the procedure would fail. For example, if the label of match is less restrictive than $\text{user} \cup \text{pwd} \cup \text{db} \cup \{\text{chkr}: \emptyset\}$, then it is not satisfied the constraints.

$$\begin{aligned}
 i, \text{match}, L_1, L_2 &= \text{user} \cup \text{pwd} \cup \text{db} \cup \{\text{chkr}: \emptyset\} \\
 L_d &= \text{user} \cup \text{pwd} \cup \text{db}
 \end{aligned}$$

Figure 25. The constraint solution of password checking example.

In contrast, our model divides the procedure into two tools shown in Figure 26: password checker and string matcher. The principal client invokes the tool password checker with four parameters: **username** and **password** provided by the client, **database usernames**, and **database passwords**. The tool password checker would invoke tool string matcher to verify whether the input username and password are consistent to the record in the database **db**.

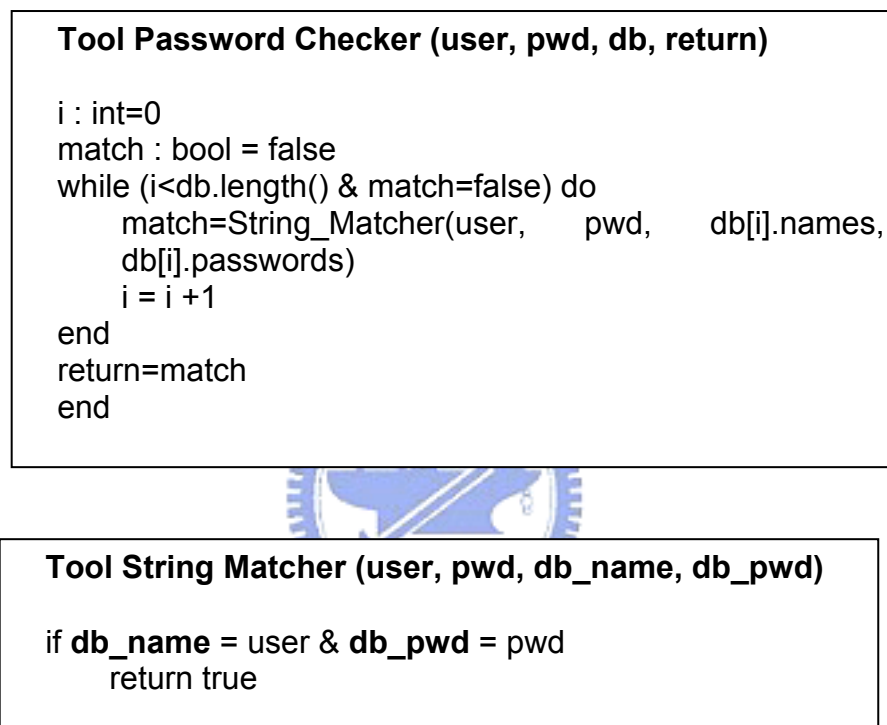


Figure 26. Checking password example performed by our model.

As mention above, we allow principals to specify declassification in the signature of tool. Thus principal password checker can specify declassification on the tool string which drops the level of match result to L, such that principal can get the information of match result.

The static check of procedure in decentralized label model is the similar with the typing rules we described in chapter 5, where labels of variables in the body like if and while should reflect the label of the guard of statement. For example, the variable

match in the if and while statements in this example would reflect the labels of L_1 and L_2 , respectively.

Tool Password Checker

	constraints
match = F	$\perp \leq \text{match}$
while	$L_1 \leq \text{match}$
match = string_matcher()	$i \cup \text{user} \cup \text{pwd} \cup \text{db} \leq$
match	
i = i+1	$L_1 \leq i$
return = match	$\text{match} \leq \text{return}$

Figure 27. Type inference and constraints generation of tool password checker using type inference algorithm.

Tool String Matcher

	constraints
return	<u>user</u> \cup <u>pwd</u> \cup <u>db</u> \leq <u>return</u>

Figure 28. Type inference and constraints generation of tool string matcher using our type inference algorithm.

We can use type inference algorithm described in chapter 5 to inference type and the constraints of a program phrase. According to recursive call of the type inference algorithm, we can get inference result of tool password checker and string matcher which is showed in Figure 27 and 28, respectively. After putting together the constraints we know that the label of **match** should be higher than the join of **username**, **password**, and **db**. The value of **match** can be dropped to lowest level in tool string matcher because the declassification specified by checker, such that value of match can flow to result.

Label Checking Rules

The process of verifying programs in decentralized label model involves two steps: first, basic block labels are propagated; then each individual statement is verified in the context of the basic block that contains it. For example, considering the statement **while** e

do c. First the label of block while should be invented which denoted as L . Then the statement c is verified as $\underline{c} \cup \underline{L}$ which \underline{c} represents the type of statement c .

The label checking rules of our model are similar with Denning and Denning [Den77] which are also similar with decentralized label model. Assuming that the statement while e do c is verified as τ and the type of statement c is verified as τ' . Then type τ' must be more restrictive than type τ , denoted as $\tau \leq \tau'$. And the upper bound of τ and τ' is the less restrictive constraint satisfied the inequality, which is similar with decentralized label model.

Assignment

Considering an example of assignment $v := e$, where v is a variable with type \underline{v} and e is an expression with typed e . In decentralized label model, this assignment is legal if $\underline{e} \cup \underline{B} \subseteq \underline{v}$ where B represents the label for the basic blocks containing this assignment. For example, consider if “ e then c ”, the whole if statement is a basic block. This implies that \underline{v} should be more restrictive than \underline{e} and \underline{B} and vice versa.

Applying to the typing rule in our model, \underline{v} should be more restrictive than \underline{e} according to the rule (assign) and rule (assign'). And v should be more restrictive than B according to the rules (If) and (If'), or (While) and (While'). Thus we can see that the constraints of decentralized label model are the same as the typing rules in our model, respectively.

If and While

Assuming that e is a legal Boolean expression with label \underline{e} , and c is a statement with label \underline{c} . Then the statement "if e then c " is legal in decentralized label model if the label of c is more restrictive than $\underline{e} \cup \underline{B}$ where \underline{B} represents the label for the basic block containing the statement. The case of "while e do c " is similar as the case "if e then c ".

In contrast, in our model, the statement "if e then c " is legal if the label of c is more restrictive than whole statement, which implies the label of c is more restrictive than e and the basic block B . The situation is similar to the case "**while e do c** ".

Authority and Revocation

In the ML model, a procedure can execute with some authority which is granted through the principal hierarchy or granted to have right for the caller of the procedure.

Whether a procedure has authority granted by a principal is tested according to the statement "if_acts_for(P_1 , P_2) then S_1 [else S_2] end". In this statement, P_2 corresponds to a principal in the principal hierarchy and P_1 corresponds to the current procedure. If the test succeeds then S_1 is executed, or S_2 is executed. Assuming the underlying can halt a running process, it is also possible to revoke the authority of a procedure by changing the principal hierarchy.

Our model does not allow a principal to assume another principal's privilege but allows a tool to grant authority for another principal. The acts-for relation in decentralized label model implies that if a procedure acts-for a principal, then it would have right to access all data belonging to the principal. Instead of granting all access right to a principal, our model allows granting partial authority to a principal on a per-tool basis. If a principal allows the authority in certain range in his own lattice definition, then the procedure has access right of data in the range granted by the principal. Thus it is possible for a tool to grant authority to a procedure for access all data in our model.

Declassification

Declassification in decentralized label model depends on the acts-for relation. Considering the password checking example above, declassification is needed to return the match result of password checking to client. Their model allows explicit declassification in the procedure, and the declassification is legal as long as removed readers belong to the principal whom the procedure acts-for. However, declassification specification which is not separate from program logic may incur policy management complexity greatly.

In our model, declassification is specified by each individual principal on signature of tools which separates programming logic and declassification specification, makes policy management clear. Declassification is legal in our model if it does not exceed the range of the authorized range.

Chapter 7 System Design and Implementation

In this section we outline the design of our system and some details about implementation. Our system is an information sharing system constructed in decentralized environment which equipped capability of policy mechanism for users to control information flow. The system provides a consistent approach to specify policy on both data and tools such that users can control the information flow as they want.

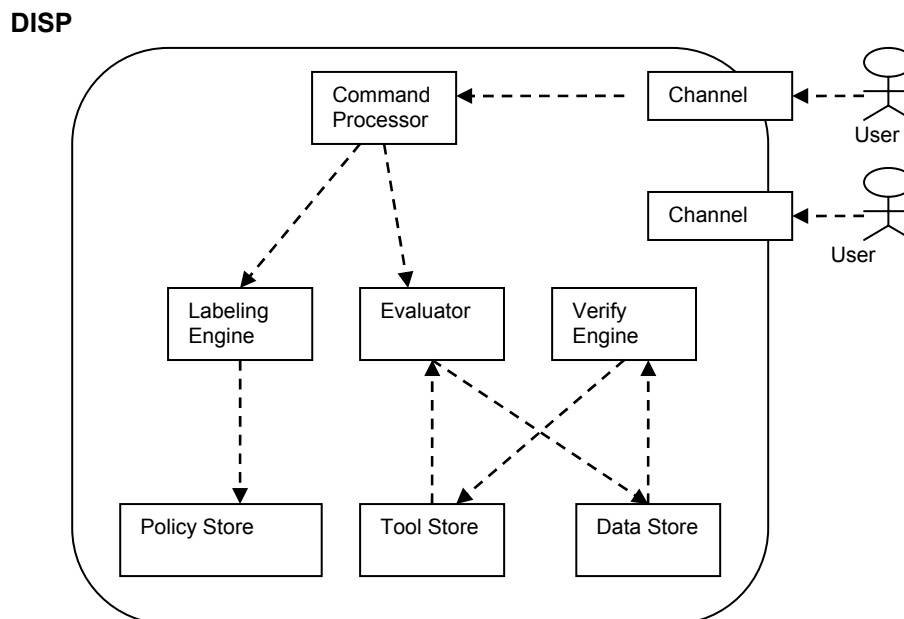


Figure 29. Overview of our system implementation.

Our system contains these basic components:

- *Realm, Channel:* In our system, data, tool and their labels are stored in a realm. And a channel is the medium for a user connect to a realm. When a person wants to use our system, he must connect to the channel and then sends his requests through the channel which he has connected
- *Command processor:* A command processor is an interface which receives commands from users and then sends them into our internal system to processing.
- *Data store:* A date store unit stores data and their labels.

- *Tool store and management*: A tool store unit stores the tools both system build-in and created by users, and also manages these tools for user to add/ delete, and modify. A tool store also stores and manages labels of tools which specifying the signature of tools.
- *Policy store and Labeling engine*: A policy stores predicates defined by users and a labeling engine attaches labels on data in data store according to policy store
- *Evaluator*: An evaluator processes using tools to process data; this unit causes the information flow.
- *Verify engine*: A verifier verifies the flow of tools and generates labels and check whether input parameters satisfy the constraints of the tool when it is called.

Here we show how our system process uses' command in some scenario cases:

Creating tool and importing data: Tools can be creates by users, a tool should be verified before it is called. Importing data can be done by a tool which produces new data. If a user wants to import new data into the realm, he can only use a tool to import. The label of new data is also determined by the tool.

Data processing: A user wants to process data using tools, and then he can send his command to command processor through the channel. The command processor parses the users' commands. Before calling tools to do operations, the verify engine would check whether the input data conform the tool's input parameter's type. If type match succeeds, then the tool processes the data which cause information flow. If type match fails, then the command also results in fail.

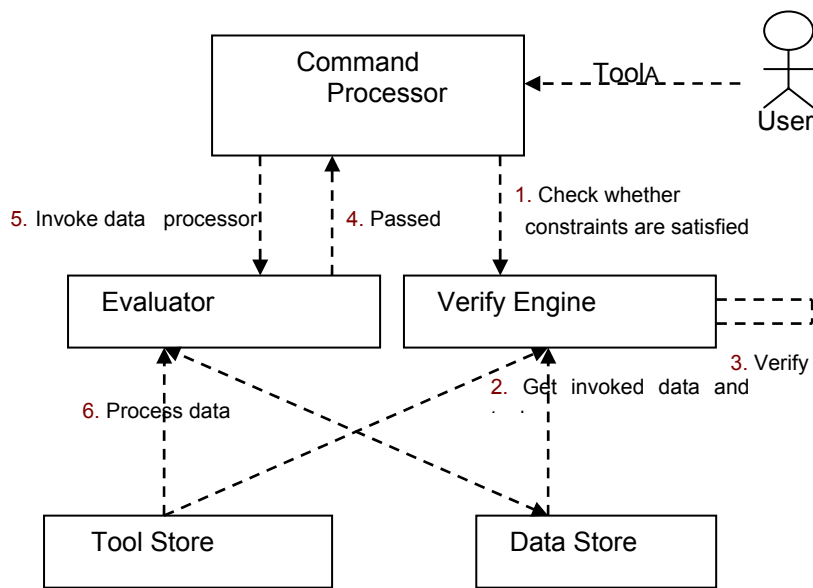


Figure 30. The flow path of data processing.

Policy specification of data: Our system uses a logic-based approach to specify policy on data. A policy can be seen as a program which contains a set of facts and a set of rules. A labeling engine can always decide what kind of label should be attached according to policy store.

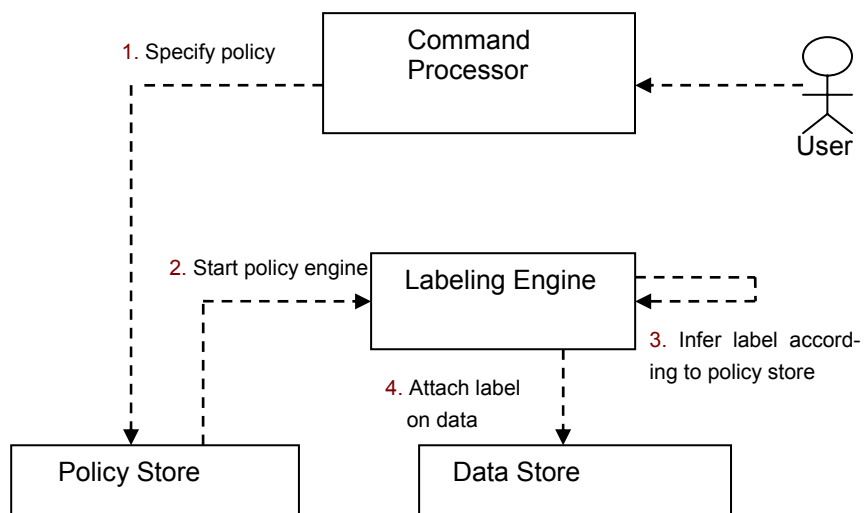


Figure 31. The flow path of policy specification.

Declassification on tool: After a user creating a tool to our system, the user can present what the tool does. For example, a statistician may create a tool which making statistics. After analyzed by verify engine, the tool would be attached a label in a document form and other users can know the effect would happen after using the tool. After understanding the document completely, a doctor can declassify the diagnosis record for this tool such that the diagnosis record only leak outside for making statistic record.

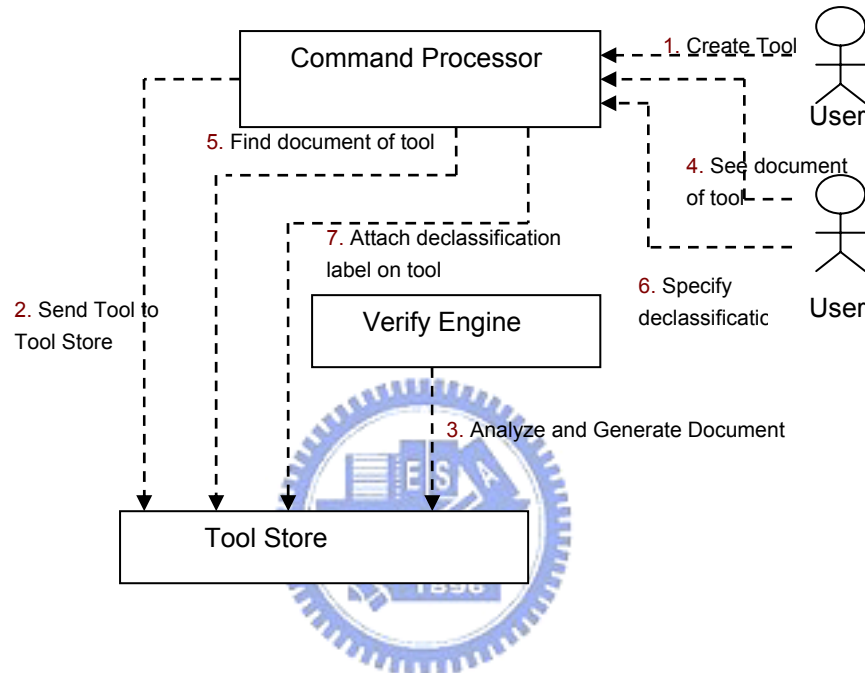


Figure 32. The flow path of declassification on a tool.

We have implemented our model using Prolog, which is suitable for type checking and inference. In particular, it is easy for each user to specify policy on data and other users by a simple program in Prolog predicates.

Chapter 8 Related Work

Andrew. C. Myers and Barbara Liskov developed many techniques in the area of decentralized information flow. They have improved traditional information flow model in decentralized settings. [Mye97] allows collaborative computation by mutually distrustful principals. Users can specify personal policy without traditional rigid multi-level constraints. [And 98] retains the advantages and extends it by safer relabelings that [Mye97] does not permit. The relabeling work depends on the acts-for relations. Therefore they incorporate the principal hierarchy into account management. The model also defines a rule for static checking that checks that a rule is both sound and complete: it allows all and only safe relabelings.

However, these models are quite powerful and too dangerous to use since the acts-for relation relies on mutual trusting among entities. Declassification in their work also depends on the acts-for relation which is not trivial to manage for a given policy requirement. Also, without a consistent approach to policy specification, declassification is not easily to handle in decentralized label model [Zan04]. For example, the examples in the ML model require mixing access control logics with program logics, making security policy enforcement and validate difficult. In contrast, our model separate security related specification from program logic and thus ease the management burden.

When it comes to control information flow with fine granularity at programming language level, [And99] provides a language called JFow, which extends their earlier work and Java language as adding statically-checked information flow annotations. The language they propose provides more flexibility by label polymorphism, run-time label checking, and type inference [And00].

[Fer97] introduces a form of dynamically-checked declassification through special waivers to strict flow checking. Some of the need for declassification in their framework would be avoided with fine-grained static analysis because waivers are applied dynamically and mention specific data objects, they seem likely to have administrative and run-time overheads. Unlike traditional centralized setting, one key advantage of their model structure is that it is decentralize; principals in the system do not need to trust the declassification of other principals. However their work also focuses on who

can access an object and faces similar complexity issue when managing large volume of objects.



Chapter 9 Conclusions

This thesis is motivated by the desire to provide a secure information sharing model with a more flexible decentralized policy mechanism than existing models. The proposed policy mechanism improves existing models and meets particular requirements of users. The limitations of existing models are that they either lack supporting declassification well or lack a consistent approach to specify policies when the volume of data is large. Many mechanisms allow acts-for relation which can make declassification risky. Policies of these models relies on per-object labeling, it would have a bottleneck when volume is large. This drawback also introduces the policy management problem. That is, when the volume of data grows, it is more likely for users to make mistakes.

The basic idea of our model is attaching labels both on data and other principals. A label represents a security level correspond to a lattice defined by a principal himself. This consistent approach improves the decentralized label model and also possesses the security properties as traditional information flow models. This model supports principals to specify individual policies, makes information flow policy management as a decentralized way.

In summary, our model contributes for decentralized information sharing as follows:

- *Flexible declassification.* Our model makes the declassification in a conditional way and avoids the possibility of unintended operations of a principal who acts for another.
- *Simplified policy management.* Our model also simplifies policy management with consistent approach to specify policies on data and tools.
- *Separation of policy specification from programming logic.* Our model allows users to specify policy for each of tools in terms of their *interfaces* but not the program body.
- *Ability to handle large volume of data.* Our rule-based approach for data labeling is simpler and easier to handle when the volume of data is high.
- *Focus on how to share and declassify.* The main difference between existing works and ours is that our model concerns more about how to share and declassify information instead of who can share and declassify information by flexible concerns of tool manipulation.

We have implemented the decentralized model by a rule-based approach, makes the information flow policies mechanism meeting the real world requirements more.

References

- [Mye97] Andrew C. Myers, Barbara Liskov, "A Decentralized Model for Information Flow Control", ACM SIGOPS Operating Systems pages.129 - 142, December 1997.
- [Mye98] Andrew C. Myers, Barbara Liskov, "Complete, Safe Information Flow with Decentralized Labels", IEEE Symposium on Security and Privacy pages 186-197, May 1998.
- [M 01] M. Satyanarayanan "Preserving Privacy in Environments with Location-Based Applications" IEEE Pervasive Computing, vol. 2, no. 1, pages 56-64, March 2003
- [Jef 01] Jeffrey Hightower Gaetano Borriello "A Survey of Context-Aware Mobile Computing Research", Technical Report, Dartmouth College, November 2001
- [San 92] Ravi's Sandhu "The typed access matrix model." In Proceedings of the Eleventh IEEE Symposium on Security and Privacy (SSP'92), pages 122-136, 1992.
- [Rav 93] Ravi's Sandhu "Lattice-based access control models", IEEE Computer Society pages 9-19, November, 1993
- [Woo 93] Woo and Simon S. Lam "Authorization in Distributed Systems: A New Approach", Journal of Computer Security pages 107-136, 1993
- [Mic 01] Michael J. Covington, Prahlad Fogla, Zhiyuan Zhan, Mustaque Ahamad "A Context-Aware Security Architecture for Emerging Applications", 18th Annual Computer Security Applications Conference, December 2002
- [Han04] Hannover, Heraklion, Linkoping, Naples, St-Gallen, Turin, Zurich, Enigmatec "Rule-based Policy Specification: State of the Art and Future Work", Project deliverable D1, Working Group I2, EU NoE REVERSE, September 2004.
- [Vol97] Dennis Volpano and Geoffrey Smith "A Type-Based Approach to Program Security" Proceedings of TAPSOFT '97, Colloquium on Formal Approaches in Software Engineering, pages 14-18 April, 1997.
- [Vol96] Dennis Volpano, Geoffrey Smith, and Cynthia Irvine "A Sound Type System For Secure Flow Analysis", Journal of Computer Security pages 1-20, Jul 1996

- [Dav03] Tom Davis and Henry A. Waxman, "File-Sharing Programs and Peer-to-Peer Networks Privacy and Security Risks", United States House of Representatives Committee on Government Reform – Staff Report Prepared for Rep. T, May 2003.
- [Zan04] Steve Zancewic, "Challenges for Information-flow Security" In Proceedings of the 1st International Workshop on the Programming Language Interference and Dependence, 2004.
- [Sam00] Sameer Ajmani, "A Trusted Execution Platform for Multiparty Computation" Masters thesis, Massachusetts Institute of Technology, July 2000.
- [Fer97] Ferrari. E., Samarati.P., Bertino.E., and Jajodias.S." Providing flexibility in information flow control for object-oriented systems". In Proc. IEEE Symposium on Security and Privacy, pages. 130–140.
- [Zda01] S. Zdancewic, L. Zheng, N. Nystrom, and A. C. Myers." Untrusted hosts and confidentiality: Secure program partitioning." In Proc. 18th ACM Symp. on Operating System Principles , pages 1–14, Banff, Canada, Oct. 2001.
- [Clk87] David D. Clark and David R. Wilson. "A comparison of commercial and military computer security policies". In Proceedings of the 1987 IEEE Symposium on Security and Privacy (SSP'87), pages 184–195, April 1987.
- [LW88] Frederick H. Lochowsky and Carson C. Woo. "Role-based security in data base management systems", In Landwehr Ed., editor, Database Security: Status and Prospects, pages 209–222, Amsterdam, The Netherlands, 1988.
- [Bel04] Andr'as Belokosztolszki "Role-based access control policy administration" Technical Report Number 586, Computer Laboratory University of Cambridge , March 2004.
- [Blp75] Bell, D.E. and L. LaPadula, Secure Computer System: Unified Exposition and Multics Interpretation. 1975
- [Bib77] Biba, K., Integrity Considerations for SecureComputer Systems. 1977, ESD/AFSC, Hanscom AFB: Bedford, MA.
- [Den77] Dorothy E. Denning and Peter. J. Denning, "Certification of programs for Secure Information Flow", Communications of the ACM 20,7 pages 504- 513, 1977.
- [Den76] Dorothy. E. Denning. "A lattice model of secure information flow". Communications of the ACM pages 236-243, May 1976.
- [Gog82] Goguen, J.A. and J. Meseguer. "Security Policies and Security Models." in Proceedings of IEEE, 1982

- [Xu04] Y. Xu, M. Lewis, K. Sycara, and P. Scerri, “Information Sharing in Large Scale Teams”, AAMAS'04 Workshop on Challenges in Coordination of Large Scale Multi-Agent Systems, 2004.
- [Gaw03] Deiter Gawlick Shailendra Mishra, “Information sharing with the oracle database”, Proceedings of the 2nd international workshop on Distributed event-based systems, pages 1-6, 2003.
- [Jap00] Tullio Jappelli, Marco Pagano, “Information Sharing in Credit Markets: A Survey”, Centre for Studies in Economics and Finance, Mar 2000.
- [Li03] Peng Li, Yun Mao, and Steve Zdancewic. “Information Integrity Policies.” In Proceedings of the Workshop on Formal Aspects in Security & Trust (FAST), September 2003.
- [Sun] Sun Microsystems <http://www.sun.com>
- [Math] MathWorld <http://mathworld.wolfram.com/>
- [OASIS] OASIS <http://www.oasis-open.org/>
- [CDT] CDT (Center for Democracy and Technology) <http://www.cdt.org/>
- [HIPAA] HIPAA (The Health Insurance Portability and Accountability Act) <http://www.hipaa.org/4>.
- [PHI] National Standards to Protect the Privacy of Personal Health Information <http://www.hhs.gov/ocr/hipaa/finalreg.html>
- [ASS] HIPAA Administrative Simplification – Security <http://www.cms.hhs.gov/hipaa/hipaa2/regulations/security/default.asp>
- [Appel] Appel (A P3P Preference Exchange Language) <http://www.w3.org/TR/P3P-preferences/#P3Ppolicies>