Information Flow Control for Cloud Environments

Shih-Chien Chou

Department of Computer Science and Information Engineering
National Dong Hwa University, Taiwan
Email: scchou [AT] mail.ndhu.edu.tw

ABSTRACT--- ***Information flow control (IFC) on cloud environments is substantially affected by the features of multi-tenant and virtualization. For example, if multiple cloud applications executes on a cloud (this is the feature of multi-tenant), the information of one or more cloud applications may be intercepted by others. As another example, when the storage units assigned to a cloud application are re-assigned to others (this is caused by virtualization), the information of the original application stored in the storage units may be leaked to others. To solve the problems, we proposes a two-layered IFC model and a flushing function. The upper layer of the model isolates information of different cloud applications to prevent possible interception. The lower layer controls information flows in a cloud application to prevent information leakage. The flushing function flushes information in a storage unit when it is re-assigned to another cloud application. This prevents an application to obtain the information belonging to other ones.***

Keywords--- information security, information flow control, cloud environment, cloud application, cloud information flow control

1. INTRODUCTION

Information security is crucial in every computer system. In general, the security ensures that information sent by a sender is securely received by a receiver. Moreover, the information being sent should not be intercepted (or leaked), changed, or corrupted during the sending process. To ensure the security, techniques such as cryptography, authentication, and access control have been developed. However, even if the information is received securely by the receiver, none can ensure that the information will not be leaked. The leakage may be induced by un-honest or absent-minded receivers. An absent-minded receiver may allow unauthorized persons or software to access sensitive information. Even if the receiver is honest and not absent-minded, information may be still leaked by applications executed in the receiver. For example, if an application output sensitive information to devices or files, the information may be accessed by unauthorized persons or software. To prevent this, information flow control (IFC) technique can be applied. The technique monitors information flows during the execution of an application and aborts the application if its output may leak sensitive information.

In the early days, IFC is applied in a single system. In the recent years, IFC has been applied to operating systems [1-3], web services [4-7], and even cloud applications [8-11]. Accordingly, IFC is essential in information security. When applying IFC on the cloud, every application executing in the cloud should be embedded with an IFC model. If IFC models executing on the cloud are incompatible, information exchange among applications may induce problems. In addition, according to the feature of *multi-tenant*, multiple applications can be executed in the same machines. In this case, information of an application may be leaked to other ones. Moreover, *virtualization* allows an application to be executed in different machines and used different storage units at different time during its execution. If the storage units assigned to a cloud application are re-assigned to others, the information of the original application stored in the storage units may be leaked to others. According to the description above, IFC models for a single system cannot ensure no information leakage because of the features of multi-tenant and virtualization. We developed a new IFC model for applications on the cloud. We named is CldIFC (cloud IFC). When designing CldIFC, we assumes that cloud providers are honest. This paper proposes CldIFC, which offers the following features:

1. It offers functions to prevent information leakage among applications.
2. It offers communication protocol to facilitate information exchange among incompatible IFC models embedded in cooperating applications.
3. It offers functions to prevent information leakage within an application.
4. It offers functions to prevent information leakage when storage units are re-assigned.

2. RELATED WORK

Mandatory access control (MAC), such as the model of Bell&Lapadula [12] and the lattice model [13-14] had long been used. According to its restriction, it offers strong protection on information. However, the strong restriction was criticized. To loosen the restriction, other IFC models have been developed, such as those based on decentralized label

(DIFC) [15], and those based on RBAC (role-based access control) [16-17]. Below we describe some important IFC researches.

In the lattice model [13-14], a lattice is defined as (SC, $\rightarrow$, $\oplus$), in which "SC" is the set of security class, "$\rightarrow$" is the can flow indicator (information can only follow the arrow to flow), and "$\oplus$" is the join operator. In the (DIFC) model [15], sensitive information is associated with a label. A label is composed of one or more policies. A policy is composed of one owner and zero or more readers. If a label possesses more than one policies, every policy should be satisfied. RBAC [18] is also applied to IFC, in which RBAC decides the authorization of a user according to the role he/she plays.

The models introduced above control information within a software system. Many of than were developed a decade or more ago. However, things developed in the old days do not means useless in the current days. For example, the models Flume [1], Laminar [2], and DStar [3] apply DIFC to track the information within an operating system. They use secrecy tag to control reading and integrity tag to control writing.

On the cloud environment, the model in [9] applies the Chinese Wall model [19] to control the information flow in the IaaS level. It places cloud machines in different sensitive area and defines rules to control inter-area information flows. The model in [10] control information flows in the SaaS level. It applies DIFC to control information flows among services. However, the affect of multi-tenant and virtualization are not taken into consideration. The article [20] emphasizes the importance of information on the cloud and makes a strange assumption that DIFC must be useful in IFC on the cloud. The article also mentions separation and tracking. However, we cannot identify a useful model in the article.

3. CldIFC

According to our survey, no available IFC model can solve the problems mentioned in section 1. We thus developed a new IFC model CldIFC for applications on the cloud. The design philosophy is borrowed from CertiKOS [21]. In CertiKOS, the authors assume that cloud providers, hypervisors, and other cloud applications are all untrusted. To prevent information leakage, the model embedded the trusted CretiKOS in the hypervisor. CertiKOS ensures that information of an application is accessed by the application only. If an application intends to access information not belonging to it, the access will be denied. Fig. 1 depicts the architecture of CertiKOS. It is located between untrusted applications (software) and hardware to limit the information that can be access by an application. This prevents illegal access.

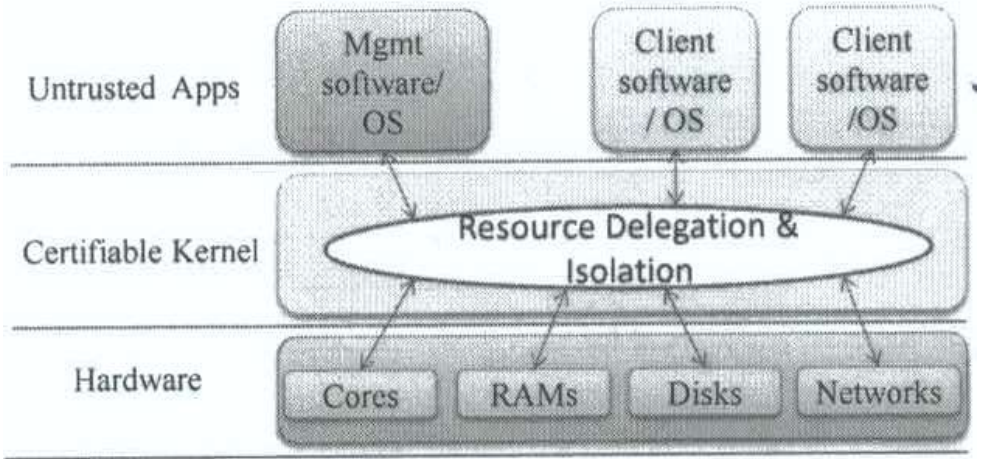

Fig. 1. CertiKOS architecture (adapted from the article of Gu et al [21])

As described above, CertiKOS ensures that information of an application is accessed by the application only. CertiKOS achieves this by separating applications from information, recording the belonging relationships between applications and information, and allowing an application to access its information only. The approach of CertiKOS indeed ensures legal access of information. However, in the case of cooperating applications (such as producers and consumers), different applications may share the same information. CertiKOS does not allow information sharing for cooperating applications. The second shortcoming of CertiKOS is that it does not offers IFC function to ensure secure information flows within an application. The third shortcoming is that CerticKOS cannot ensure that information will not be leaked when storage units are re-assigned. Accordingly, CldIFC is designed based on the concept of CertiKOS but offer improvement. CldIFC is a two-layered architecture described below:

1. CldIFC separates applications from information to ensure secure information access like CertiKOS. However, CldIFC allows cooperating software. Cooperating applications can share information but non-cooperating ones not. The separation is a function of the upper layer of CldIFC.
2. CldIFC controls information flows within cloud application during execution. It is a function of the lower layer of

CldIFC, which ensures that no statement will leak information during execution.

3. CldIFC offers protocol to communicate incompatible IFC models embedded in cooperating applications. It is a function of the upper layer model.
4. CldIFC cooperates with the hypervisor to handle the problems induced by virtualization. When the hypervisor re-assigns one or more storage units, CldIFC requires the hypervisor to flush the contents of the units. This function is independent (i.e., does not belong to CldIFC). According to the above description, CldIFC should closely cooperate with the hypervisor. Nevertheless, since the hypervisor of a cloud is offered by the cloud provider, our research is thus limited to simulation. However, if the flushing function is proved to be valuable, we hope that the function can become a standard for cloud environment. Below we describe CldIFC in details.

As described above, CldIFC is composed of an upper and a lower layers. The former handles information sharing among cooperating applications. It also offers protocol to communicate incompatible IFC models embedded in cooperating applications. The lower layer model controls information flows within an application, which prevents output information leaking to persons or files. In addition, CldIFC closely cooperates with the hypervisor to do the necessary flushing operations. Since the cooperation between CldIFC and the hypervisor can only be simulated, we focus on the discussion of CldIFC. In our research, we use lattices to develop the two layers of models. We first explain why lattice is selected to use.

We identified that many IFC model embedded lattices. The early IFC model developed by Bell&LaPadula [12] is generalized into the MAC (mandatory access control) developed by Denning [13]. The architecture of MAC is a lattice. It is composed of a minimum node, a maximum node, and other nodes. In a lattice, every node is a security class (i.e., security level). To allow a piece of information in a node to flow to another node, a "can flow" arrow should exist between the nodes. Since it is unnecessary to exist an arrow between every pair of nodes, a lattice is a partial ordered data structure. The lattice of Danning is depicted in Fig. 2, in which x, y, z, can be considered access rights, and a set composed of x, y, or z is a security class.

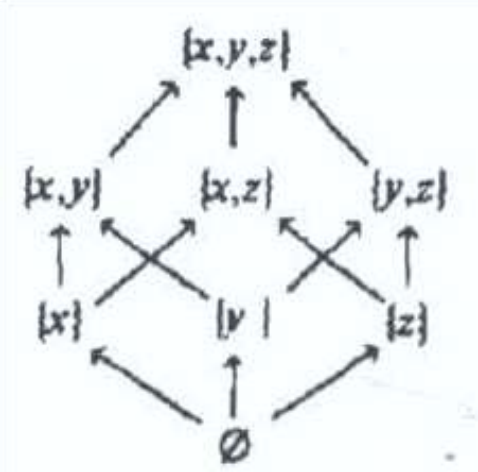

Fig. 2. Lattice (adapted from the article of Danning [13])

In the lattice of Danning, the sensitivity of information in the upper nodes is higher. Sometimes, this is unacceptable. For example, an advertisement is non-sensitive, which is in the "ϕ" node in Fig.2. If it is sent to a manager, which is in the "{x}" node in Fig. 2, the sensitivity of the advertisement becomes higher. This is meaningless. Moreover, information in a lattice can only be flowed to more sensitive location (i.e., following the arrows to flow). This causes the lattice model becomes restricted and limits its flexibility. In our viewpoint, as long as information is not output, neither persons nor software can access the information (we exclude virus, worms, and Trojan horses because they belong to other research areas). According to this observation, the restriction of the lattice model (i.e., MAC) should be adjusted. In the past two decades, many IFC models have been developed. In the models, the RBAC-based models and the label-based models are widely applied (RBAC is the abbreviation of role-based access control and DIFC [15] is the most famous label-based model). As described before, many IFC model embeds lattices. Below we describe the lattices of RBAC-based model and the label-based model.

In a RBAC model, users play roles and roles possess permissions. With this, a role can be considered as "a set of permission". The sets (i.e., roles) can be constructed into a role hierarchy. In the architecture, if the permissions of $role_1$ include all permissions of $role_2$, $role_1$ dominates $role_2$. The role hierarchy can be transformed into a lattice following the steps below: (1) regarding the top of the role hierarchy as the maximum node, (2) reverse the direction of all the arrows in the hierarchy, and (3) add a empty set as the minimum node of the lattice and place arrows from the minimum to every bottom node of the role hierarchy. After the transformation, the role hierarchy becomes a lattice, in which arrows are the "can flow" relationships. Moreover, since it is unnecessary to consist of an inclusion relationship between every pair

of role permissions, the transformed architecture is partial ordered. As to sessions in RBAC, we suppose that software will use different lattice under different situations (i.e., an application is in different session under different situation). Fig. 3. depicts a transformation example.

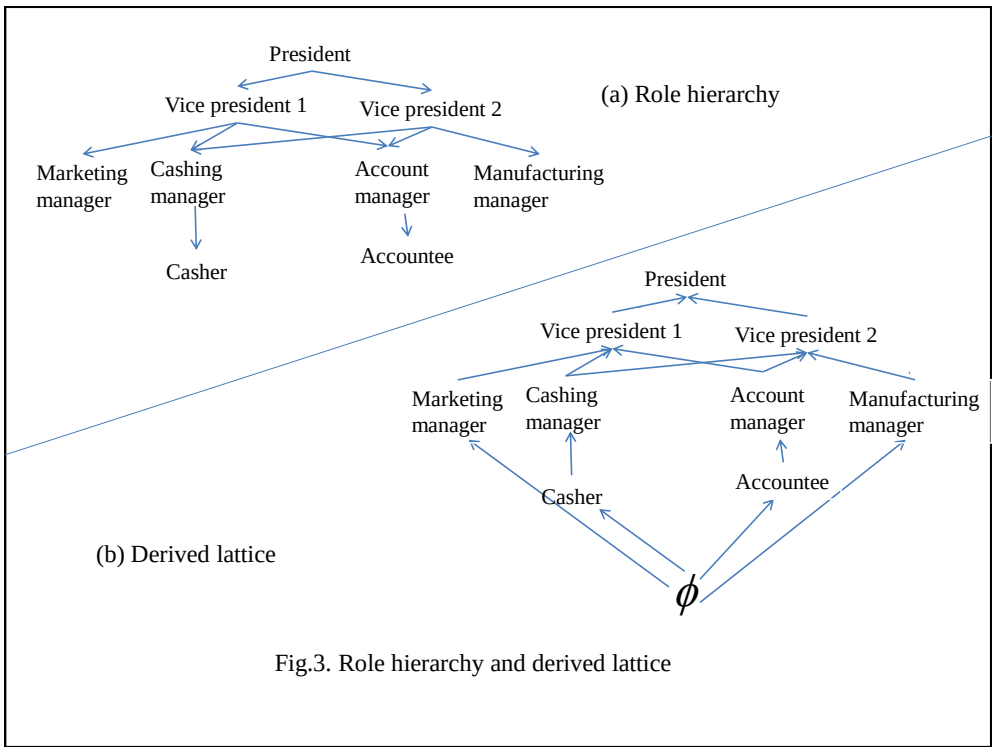

Fig.3. Role hierarchy and derived lattice

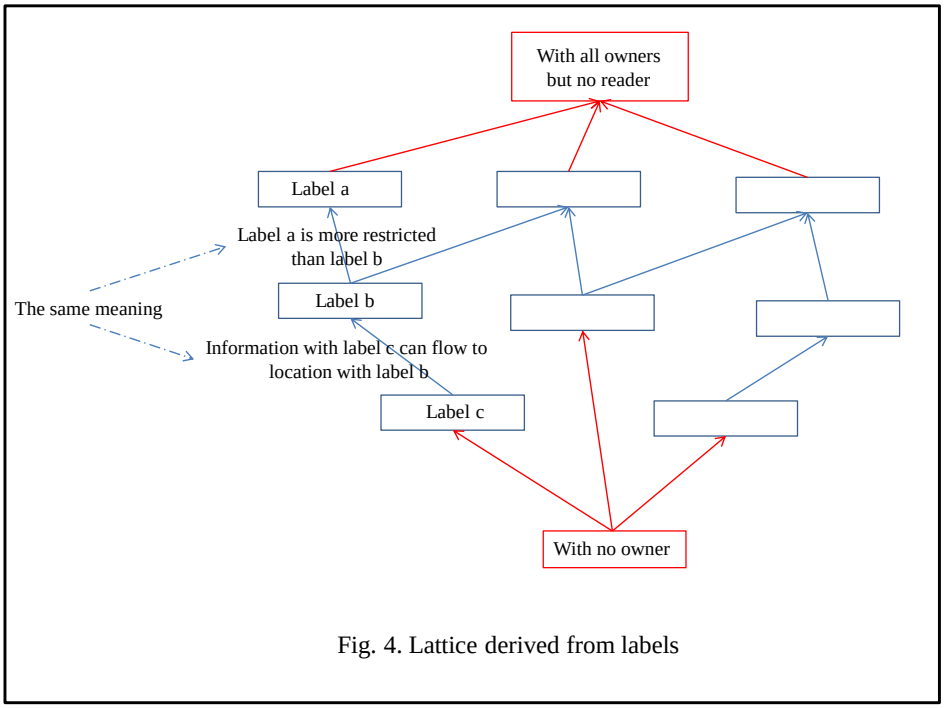

Fig. 4. Lattice derived from labels

As to deriving lattices from label-based models, we use DIFC model [15] for the explanation. In DIFC, a label is a security level. A label can be owned by zero or more owners. Each owner can allow zero or more readers to access the information the owner owns. When a label is attached to a piece of information, the information can only be accessed by the intersection of all the reader sets of owners. Moreover, whether information *A* can be flow to the destination *B* should be decided by the "more restricted than" relationships among labels. Generally, more owners and fewer readers results in more restriction. Suppose the DIFC model in a software system possesses the owner set "OW" and the reader set "RD". Then, "OW" and "RD" can establish all labels needed by the software system. Among the labels, the

"more restricted than" relationships can be used to build the "can flow" relationships. Nodes as well as the "can flow" relationships construct a partial ordered graph. In the graph, a label with no owner can be added as the minimum node, whose information can flow to every bottom label. Moreover, a label with all owners and no reader can be added as a maximum node and information attached with the most restricted label can flow to the maximum node. With this, a lattice has been established (see Fig. 4).

Since most IFC model embeds a lattice, both the two layers of CldIFC are developed based on lattice. Nevertheless, there are shortcomings for MAC, RBAC-based models, and label-based models, as described below:

1. Only control read access but not write access. Read access is necessary to prevent information leakage. As to write access, it prevents malicious information corruption. For example, if the security level of *a* is higher than *b*, *a* can read the information of *b*. In other words, the information of *b* can be written to *a*. In this case, if *b* contains garbage information, writing the information of *b* to *a* corrupts the originally useful information of *a*. With this consideration, CldIFC handles write access control.
2. Information can only follow the "can flow" link to *flow up* to the destination. This is too restricted. In our opinion, if virus, worms, and Trojan horses are not considered (they belong to other research areas), only output information may be leaked (to persons or other software systems), because information operating by a software system cannot be accessed by persons or other software systems. Under this consideration, allowing information to *flow down* following the reversed direction of a "can flow" link should be allowed if the flowing is not an output operation. However, allowing the flow should adjust the security level of the destination because it receives high security level information. The adjustment prevents information leakage when the destination is output later. We call this adjustment a "join operation" [15].

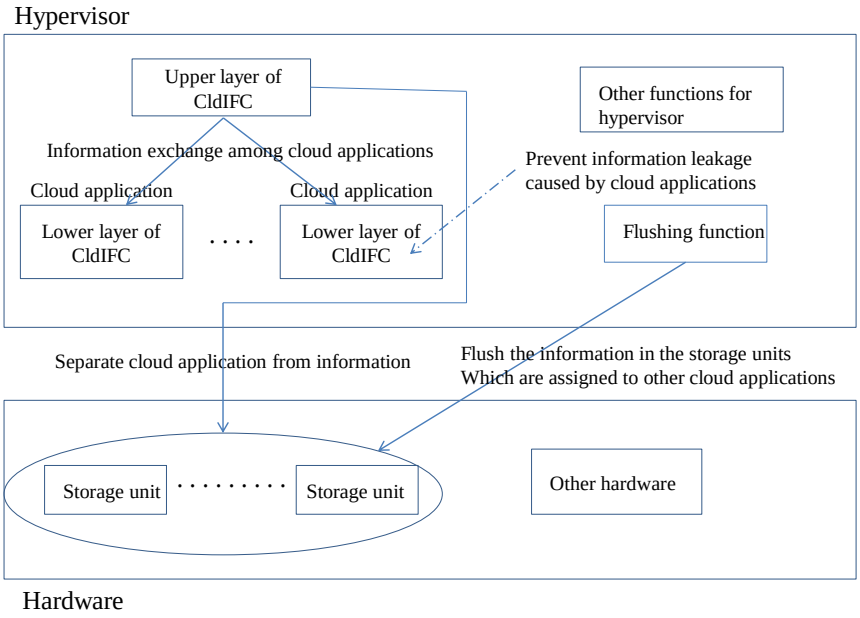

Fig. 5. The architecture of CldIFC

Having described the two problems, we describe two-layered CldIFCthe flushing function. Their architecture is depicted in Fig. 5. We apply RBAC to develop the two-layered model (since the core of RBAC is a lattice, CldIFC is also based on lattice as mentioned above). In the model, we regard a role as a set of permissions, in which a permission is defined below:

Definition 1: A permission is defined as "(ssn, sj, oj, right)", in which:

- **a.** *ssn* is a session. As will be described, sessions are different in the two layers of CldIFC.
- **b.** *sj* is a subject to access object. In the upper layer of CldIFC, a subject is a cloud application. In the lower layer, we let a subject be an object method.
- **c.** *oj* is the object to access. In the upper layer, *oj* is the information accessed by cloud applications. In the lower layer, *oj* is an object attribute.
- **d.** *right* is an access right. It can be "read" or "write". We need the "write" access right to control write access.

According to Definition 1, CldIFC is defined below (the flushing function is temporarily ignored here):

Definition 2: CldIFC is defined as "(CldIFC_U, CldIFC_L)", in which the former is the upper layer model of CldIFC and the latter the lower one. CldIFC_U will be embedded in the hypervisor to separate applications from information (the flush function is also embedded in the hypervisor). CldIFC_L will be embedded in cloud applications to control information flows. Since both layers are developed based on RBAC, they are all defined as "(USR, RLE, SSN, URA, CNT, RH)", in which:

A. USR is the set of users. In both layers, a user is a client of a cloud application.
B. RLE is the set of roles. A role is a set of permissions. Since permissions are embedded in roles, permissions need not appear in Definition 2. See Definition 1 for the definition of a permission.
C. SSN is the set of sessions. In the lower layer model, a session is the execution duration of a cloud application. In the upper layer model, a session is composed of cooperating applications. Cloud applications in the same session can exchange information. This remedies the shortcoming of CertiKOS, which cannot handle information exchange among cooperating applications.
D. URA is the set of user-role assignments. The definition of URA is the same as that in RBAC.
E. CNT is the set of constraints. The most familiar constraint may be SoD (separation of duty).
F. RH is role hierarchy. As shown in Fig. 3, RH is the primary element to produce lattices.

Below we describe the operation of CldIFC using the definitions. The lower layer of CldIFC (i.e., CldIFC_L) controls information flows within an application. As described above, only output operations may leak information. Therefore, CldIFC_L strictly controls output operations by allowing information to be output to files/devices with the same or higher security levels. As to other operations, CldIFC_L allows them to execute. However, it applies the join operation to adjust the security level of information. The adjustment prevents possible information leakage when information is output later. When executing an output operation, both the following rules should be true. In the rules, we suppose *attr* is intended to be output to the o_1. Moreover, *attr* is obtained from the set *ATT* defined as: {att_i | att_i is an attribute of an object (see Definition 1 for the term "object")}.

Rule 1: $\forall att_i \in ATT, \exists per_j = (ssn_k, o_1, att_i, "read")$

Rule 2: $\forall att_i \in ATT \wedge att_i \in o_j, \exists per_k = (ssn_k, o_j, attr, "write")$

In the rules above, Rule 1 and Rule 2 respectively control the read and write operation. Although CldIFC_L offers rules to control output operations, information output to cloud devices/files should be very careful. Perhaps cloud providers are honest. However, cloud users may be untrusted. In our opinion, cloud applications should not output sensitive information to cloud devices/files in case that the size of the information is small. Instead, it should be sent back to the client and output to the client devices/files. If mass information should be temporarily stored in cloud storage units, the storage units should be carefully controlled. Of the first, the storage units should be separated from the processing units. This facilitates separating cloud applications from information. Of the second, storage space should not be used in a virtualized manner. In other words, the storage space used by a cloud application should be the same during the execution of the application. Although our consideration will increase the security of application information, we cannot control the approaches of cloud management. We can only simulate our approach in the laboratory.

Rule 1 and Rule 2 controls output statements. As to non-output ones, they will not leak information after excluding Trojan horses, worms, and virus. Therefore, non-output statements are always allowed to execute. However, to prevent information leakage when it is output later, the join operation should be used to adjust the security level of variables used in the non-output statements. To explain the join operation, we suppose that o_1 is assigned the value *attr*. Moreover, *attr* is obtained from the set *ATT* defined as: {att_i | att_i is an attribute of an object (see Definition 1 for the term "object")}. With the assumption, the join operation is defined below:

Join 1: $\forall att_i \in ATT, add\_permission(ssn_x, o_1, att_i, "read")$

Join 2: $\forall att_i \in ATT \wedge \exists o_j \in RLE \wedge \exists per_k = (ssn_x, o_j, att_i, "write"), add\_permission(ssn_k, o_j, attr, "write")$

Join 1 adjusts the reading privilege of o_1. It allows o_1 to read all att_i. Join 2 adjusts the writing privilege of objects on *attr*. It allows all objects that possess the attribute att_i to write *attr*. *RLE* in Join 2 is defined in Definition 2.

The upper layer of CldIFC (i.e., CldIFC_U) is embedded in hypervisor to separate cloud applications from information and controls information exchange among cooperating applications. It uses sessions to recognize cooperating applications. Only applications in the same session can exchange information. Therefore, when an application intends to send information to another, CldIFC_U first checks whether they are in the same session. If the answer is positive, the

information exchange is allowed. During information exchange, a communication protocol is needed. The protocol is designed for information exchange between incompatible IFC models. In this regard, applications should offer software modules to do the necessary transformation for the communication protocol. Fig. 6 shows the protocol, in which *xform* is the abbreviation of *transform*. In the figure, the calling cloud application transforms the arguments and their sensitivity levels into those required by the communication protocol. After the parameters of the receiving cloud application receive the arguments, the receiver transforms the received values and security levels for the receiver to use. Since transferring arguments to parameters can be regarded as assignment, the transferring is always allowed (see the description of CldIFC_L). However, the join operation (i.e., Join 1 and Join 2 above) should be applied to the transferring. That is, the security level of and a parameter receiving an argument should be set the same as that of the argument. As to the return value, the processing is the same as that of the arguments.

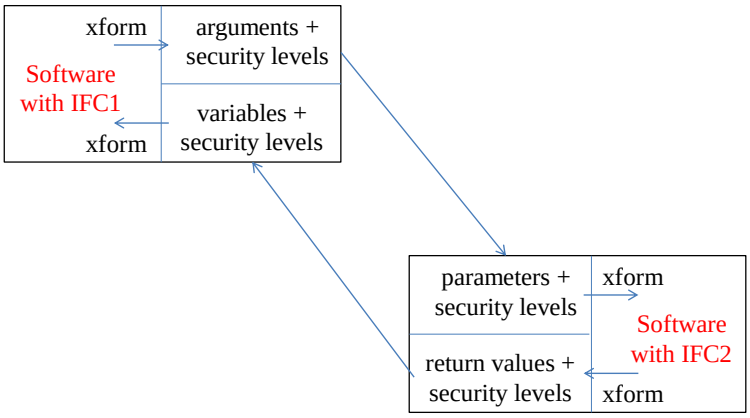

Fig. 6. Information transformation in the upper layer of CldIFC

Below we use an algorithm to describe the operations of CldIFC. The algorithm is event-driven. That is, CldIFC react to events.

Algorithm 1. The operations of CldIFC and the flushing function
Event: one or more storage units of an application are re-assigned to other applications.
React: CldIFC informs the hypervisor to flush the storage units.
Event: application ap_1 sends the argument set $\{arg_i \mid arg_i$ is an argument$\}$ to the parameter set $\{par_i \mid par_i$ is a parameter$\}$ of application ap_2, in which the numbers of arguments and parameters are the same.
React: every arg_i and par_i are managed following the steps below (the steps constitute the operations of CldIFC_U):
Step1: CldIFC checks whether ap_1 and ap_2 are in the same session. If the answer is positive, progress to the next step. Otherwise, abort both ap_1 and ap_2.
Step2: ap_1 transforms the security level of arg_i used in ap_1 into CldIFC permissions of arg_i (see Definition 1 for the definition of CldIFC permission).
Step3: par_i receives the value of arg_i and perform Join 1 and Join 2 for par_i to obtain the CldIFC permissions of par_i.
Step 4: ap_2 transforms CldIFC permissions of par_i into the security level of par_i used in ap_2.
Event: an output operation in an application embedded with CldIFC_L is required to execute.
React: CldIFC uses Rule1 and Rule 2 to check whether the output is secure. If the security is ensured, the output operation can be executed. Otherwise, the application is aborted.
Event: a non-output operation in an application embedded with CldIFC_L is required to execute.
React: execute the operation and then perform Join1 and Joint 2.

5. SIMULATION

We simulated the operation of hypervisor on the Windows system. The simulated hypervisor simulates the function of multi-tenant, virtualization, flushing storage units, and offering the functions of the upper layer of CldIFC. Machines and

storage units are also simulated. The hypervisor assumes that a computer system is composed of multiple machines (i.e., CPUs). Moreover, storage units are dynamically allocated from the Windows system. In addition to the hypervisor, machines, and storage units, software systems are also simulated. A software system is a stub. According to multi-tenant, multiple simulated software systems can exist at the same time. According to virtualization, a software system may change machines and/or storage units. As to the lower layer of CldIFC, it is embedded in the simulated software.

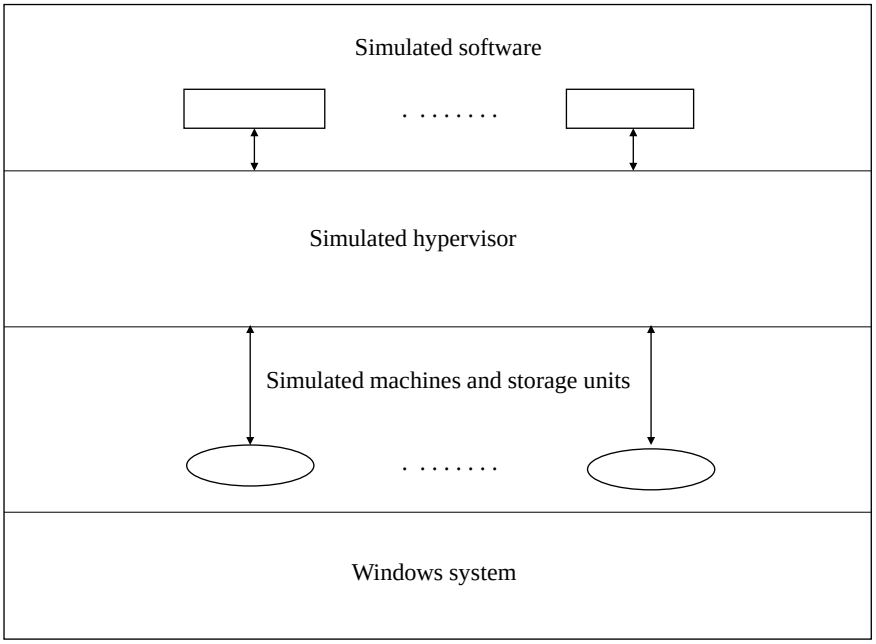

Fig. 7. The simulation architecture of CldIFC

In the simulated system, all functions of hypervisor are under control. Therefore, the functions of multi-tenant, virtualization, flushing storage units, and separating cloud applications from information are guaranteed. To obtain experiment data, we embedded different IFC models in the simulated software and injected statements that violate the CldIFC rules to the simulated software. The simulation results depict that CldIFC work well. That is, CldIFC separated applications from information and facilitates information exchange among cooperating applications. Moreover, CldIFC identified all injected non-secure indoemation flows.

6. CONCLUSION

Information flow control (IFC) prevents information leakage during the execution of a software system, in which the system may be executed in a single machine, a distributed system, or a cloud environment. Many IFC models have been developed for software systems in a single machine or a distributed system. Models are also developed for a cloud environment. When applying IFC models on a cloud environment, multi-tenant and virtualization may cause trouble, as described below:

a. According to the cloud computing feature of multi-tenant, cooperating cloud applications may execute in parallel in the same time on a cloud environment. Since cooperating cloud applications may exchange sensitive information, IFC information should also be correctly exchange to protect the sensitive information being exchanged.
b. Within a cloud application, information leakage should be prevented using an IFC model.
c. According to the cloud computing feature of virtualization, a cloud application can be executed in different machines and used different storage units during its execution. In this case, information of an application left in a storage unit should be flushed.

We developed a two-layered IFC model CldIFC (cloud IFC). It offers the following features:

1. The upper layer CldIFC model, which is CldIFC_U, uses sessions to differentiate cooperating applications from non-cooperating ones. Cooperating applications can exchange information but non-cooperating ones not.
2. CldIFC_U offers communication protocol to facilitate information exchange among incompatible IFC models embedded in cooperating applications.
3. The lower layer CldIFC model, which is CldIFC_L, prevents information leakage within an application.
4. CldIFC cooperates with the hypervisor to flush storage units when they are re-assigned. This prevent information leakage after the re-assignment of storage units.

We simulated CldIFC in our laboratory and the simulation shows that the two-layered structure of CldIFC and the flushing function works well.

7. REFERENCES

1. M. Krohn, A. Yip, M. Brodsky, and N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris, "Information Flow Control for Standard OS Abstractions", *SOSP'07*, 2007.
2. I. Roy, D. E. Porter, M. D. Bond, K. S. McKinley, and E. Witchel, "Laminar: Practical Fine-Grained Decentralized Information Flow Control", *PLDI'09*, 2009.
3. N. Zeldovich, S. Boyd-Wickizer, and D. Mazières, "Securing Distributed Systems with Information Flow Control", *NSDI '08*, pp. 293–308, 2008
4. S. –C. Chou and C. –H. Huang, "An Extended XACML Model to Ensure Secure Information Access for Web Services", *Journal of Systems and Software*, vol. 83, no. 1, pp. 77-84, 2010.
5. S. –C. Chou, "Dynamically Preventing Information Leakage for Web Services using Lattice", *5'th International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*, 2010.
6. W. She, I. –L. Yen, B. Thuraisingham, and E. Bertino, "The SCIFC Model for Information Flow Control in Web Service Composition", *2009 IEEE International Conference on Web Services*, 2009.
7. W. She, I. –L. Yen, B. Thuraisingham, and E. Bertino, "Effective and Efficient Implementation of an Information Flow Control Protocol for Service Composition", *IEEE International Conference on Service-Oriented Computing and Applications*, 2009.
8. W. She, I. -L. Yen, B. ThuraiSingham, E. Bertino, "The SCIFC Model for Information Flow Control in Web Service Composition", *2009 IEEE International Conferences on Web Services*, pp. 1-8, 2009.
9. R. Wu, G. –J., Ahn, H. Hu, and M. Singhal, "Information Flow Control in Cloud Computing", *Proceedings of the 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 2010.
10. T. Liu and Y. Zhou, "A Decentralized Information Flow Model for SaaS Application Security", *Third International Conference on Intelligent System Design and Engineering Applications*, pp. 40-43, 2013.
11. S. –C. Chou, "Controlling Information Flows in SaaS Cloud applications", *ICCIT*, 2012.
12. D. E. Bell and L. J. LaPadula, "Secure Computer Systems: Unified Exposition and Multics Interpretation", *technique report, Mitre Corp.*, Mar. 1976. http://csrc.nist.gov/publications/history/bell76.pdf
13. D. E. Denning, "A Lattice Model of Secure Information Flow", *Comm. ACM*, vol. 19, no. 5, pp. 236-243, 1976.
14. D. E. Denning and P. J. Denning, "Certification of Program for Secure Information Flow", *Comm. ACM*, vol. 20, no. 7, pp. 504-513, 1977.
15. A. Myers and B. Liskov, "Protecting Privacy using the Decentralized Label Model", *ACM Trans. Software Eng. Methodology*, vol. 9, no. 4, pp. 410-442, 2000.
16. K. Izaki, K. Tanaka, and M. Takizawa, "Information Flow Control in Role-Based Model for Distributed Objects", *8'th International Conf. Parallel and Distributed Systems*, pp. 363-370, 2001.
17. S. -C. Chou, "Embedding Role-Based Access Control Model in Object-Oriented Systems to Protect Privacy", *Journal of Systems and Software*, 71(1-2), 143-161, Apr. 2004.
18. D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-Based Access Control", *ACM Trans. Information and System Security*, vol. 4, no. 3, pp. 224-274, 2001.
19. Brewer, D.F.C., Nash, M.J., 1989. The Chinese Wall Security Policy. In: Proceedings of the 5'th IEEE Symposium on Security and Privacy, 206-214.
20. J. Bacon, D. Eyers, T. F. J. –M. Pasquier, J. Singh, I. Papagiannis, and P. Pietzuch, "Information Flow Control for Secure Cloud Computing", *IEEE Trans. Network and Service Management*, 11(1), pp. 76-89, 2014.
21. L. Gu, A. Vaynberg, B. Ford, Z. Shao, and D. Costanzo, "CertiKOS: A Certified Kernel for Secure Cloud Computing", *APSys'11*, 2011.