CONTROLLING INFORMATION FLOWS DURING SOFTWARE DEVELOPMENT

Shih-Chien Chou

Department of Computer Science and Information Engineering, National Dong Hwa University, Taiwan

ABSTRACT

Information flow control (IFC) is useful in preventing information leakage during software execution. Our survey reveals that no IFC model is applied on the entire software development process. Applying an IFC model on the entire software development process offers the following features: (1) viewpoints of all stakeholders (i.e., customers and analysts) can be included and (2) the IFC model helps correcting statements that may leak information during every development phase. In addition that no IFC model is applied to the entire software development process, we failed to identify an IFC model that can reduce runtime overhead. According to the above description, we designed a new IFC model named PrcIFC (process IFC). PrcIFC is applied on the entire software development process. Moreover, PrcIFC is disabled after software testing to reduce runtime overhead.

KEYWORDS

Software security, information flow control, software development process, reduce runtime overhead

1. INTRODUCTION

Information security covers many issues such as cryptography, authentication, and access control. *Information flow control* (*IFC*) is a branch of access control. It prevents information leakage during software execution. Information leakage occurs when persons or other software illegally obtain sensitive information from a software system. In general, persons can obtain information from files or output devices and software can obtain information from files. Information leakage may thus happen *when sensitive information is output*.

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 [8-9]. In other words, IFC has been applied to the entire Internet. IFC is thus essential in information security. We involved in the research of IFC for years and identified that no IFC model is applied on the entire software development process. Applying IFC on the entire software development process offers the following features: (1) viewpoints of all stakeholders (i.e., customers and analysts) can be included and (2) the IFC model helps correcting statements that may leak information during every development phase. We also identified that no IFC model take runtime overhead reducing into consideration, in which the overhead is induced by embedding IFC models in software. We thus designed a new IFC model to overcome the problems mentioned above. The design is based on the following considerations:

DOI : 10.5121/ijsptm.2016.5301

a. Embedding the IFC model into software specifications and design documents. Traditional models are embedded in software only, which may exclude the viewpoints of some stakeholders.
b. Executing the specifications and design documents in which the IFC model is embedded. This execution causes the IFC correctness of specifications and design documents to be checked.
c. Embedding the IFC model in the software during testing. The testing thus tests both the correctness of functions and IFC. After testing, the model is disabled to remove runtime overhead induced by IFC model.

As described in term *c* above, the IFC model proposed in this paper is disabled after testing. Nevertheless, software that passes the testing is not ensured to be correct. Therefore, the operating system should intercept software output information to ensure security (OS intercepts only output information because only output information may be leaked to persons or other software). In our research, we design the IFC model and let the OS interception as a future work. Since the model is applied in the entire software development process, we name it *PrcIFC* (process IFC). The core of PrcIFC is the same as NetIFC [10] we designed before. When designing PrcIFC, we ignored the affect of viruses and Trojan horses because they belong to other research areas. During the development of PrcIFC, we identified the following facts.

a. Since only output information may be leaked (to persons or other programs), only output statements need to be controlled. As to other statements, the *join* operation [11] should be used to adjust variable sensitivity after execution. This adjustment prevents possible information leakage when the variable is output later.
b. Information should be separated in different group for compatibility. For example, prices with the units of USD and EUR are incomparable.

According to the facts, PrcIFC controls output statements and uses join operations to adjust information sensitivity for other statements. PrcIFC presented in this paper offers the following major features:

a. PrcIFC is applied on the entire software development process. Therefore, it will not exclude the viewpoints of any stakeholders and helps correcting statements that may leak information during every phase of the development.
b. PrcIFC is removed when software is online, which remove runtime overhead.

2. RELATED WORK

Traditional lattice-based model [12-13] is a MAC (mandatory access control) which is criticized as too restricted. The decentralized label model [11, 14] uses labels to mark the security levels of variables. A label is composed of policies to be simultaneously obeyed. The model in [15] uses ACLs of objects to compute ACLs of executions.

The model Trust-Serv [16] uses state machines to dynamically choose web services at run time. It uses trust negotiation [17] to select web services that can be invoked. The kernel component to determine whether a web service can be invoked is *credential*. The model proposed in [18] uses digital credentials for negotiation. It defines strategies for negotiation policies. SCIFC [6] uses various mechanism and algorithms to make sure whether a service chain can be successfully invoked.

Flume [1] is a decentralized information flow control (DIFC) model for operating systems. It tracks information flows in a system using tags and labels. The control granularity is detailed to processes. The secrecy tags prevent information leakage and integrity tags prevent information corruption. Flume also avoids information leaked to untrusted channel. The function of Laminar [2] is similar to that of Flume.

The model proposed in [19] uses X-GTRBAC [20] to control the access of web services. X-GTRBAC can be used in heterogeneous and distributed sites. Moreover, it applies TRBAC [21] to control the factor related to time. The model in [22] uses RBAC (role-based access control) concept [23-24] to define policies of accessing a web service. It is a two-levelled mechanism. The first level checks the roles assigned to requesters and web services. The second level uses parameters as attributes and assigns permissions to the attributes. An authorized requester can invoke a web service only when it possesses the permissions to access the attributes.

The model in [25] identifies dependencies among I/O parameters of services to decide whether service invocations will leak information. The model in [26] offers functions to check intra-component information flows through the JIF language [27]. The model in [8] applied the Chinese Wall model [28] to control information flows in the IaaS level. As to services in the SaaS level, the model cannot control their information flows. In the recent survey of [9], the authors especially emphasize the importance of data privacy in a cloud computing environment. They believed that information flow control is a good solution for the problem. However, the article did not propose a concrete information flow control model.

According to our survey, no research applies IFC in the entire software development process and almost no one discusses runtime overhead.

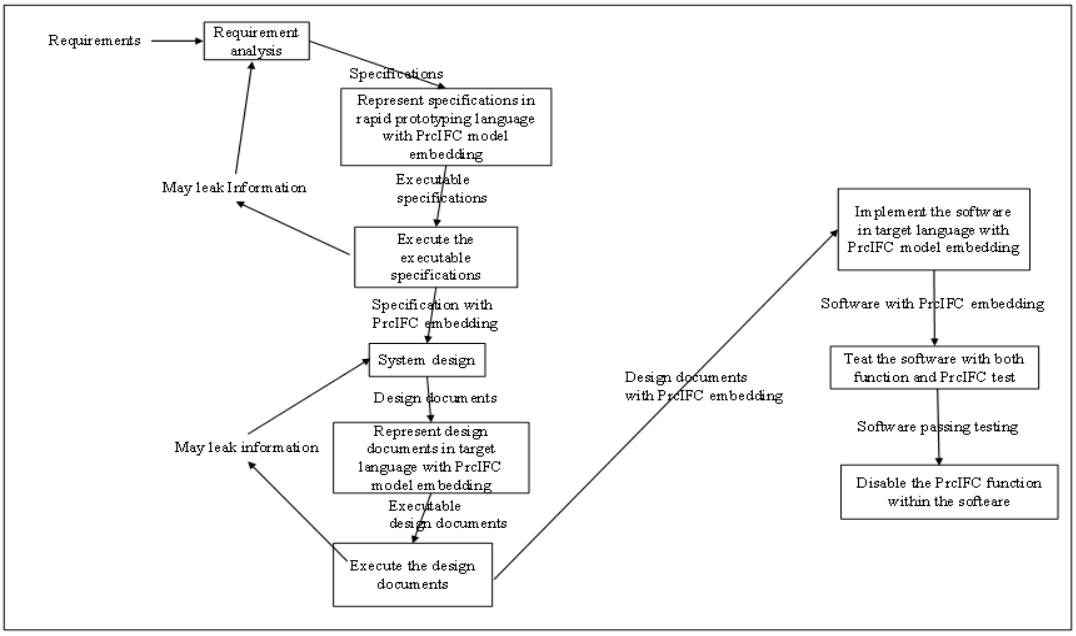

Figure 1. Design Philosophy Of Prcifc

3. PRCIFC

Figure 1 depicts the design philosophy of PrcIFC. Before describing Figure 1, we assume that software development tools, such as the UML development platform, are trusted. Therefore, PrcIFC needs not handle possible problems induced by software development tools.

Figure 1 shows that we embed PrcIFC in specifications then execute them. To execute a specification, we use our rapid prototyping language [29] to write the specification. That is, we embed PrcIFC in our prototyping language. If a specification passes the test of both functional and IFC requirements, the specification can be designed. Otherwise, the requirement should be re-analyzed. During the design, we use the target language (such as C) to write the design documents and embed PrcIFC in the documents. The documents are then executed and tested. If the design documents pass the test of both functional and IFC requirements, the design document can be implemented. Otherwise, the design activity should be repeated. During implementation, PrcIFC is embedded in the programs. The programs are then executed. If the programs pass the test of both functional and IFC requirements, the programs are released to the customers as a software system. Before the software system is online, PrcIFC is disabled to completely remove the runtime overhead. When the software is online, the OS intercepts the output of the software to check possible information leakage. However, the interception is out of the scope of this paper.

According to the description in section 1, information should be associated with a mechanism to represent its sensitivity. The *sensitivity should include a security level number and group*. Moreover, we need a join operation to adjust the sensitivity of variables in non-output statements. In this regard, we let a variable's sensitivity be "(*Sln*, *Gv*)", in which *Sln* is the security level number whereas *Gv* is the group. *We call the sensitivity a security level*. Therefore, every piece of information that should be protected is associated with a security level. The definition of security levels results in a lattice. According to the lattice, only output statements are controlled strictly and other statements applies the join operation to adjust the security level of variables. Suppose a piece of information derived from the variable set "$\{b_i \mid b_i$ is a variable associated with the security level $(Slv_{b_i}, Gv_{b_i})\}$" is outputting to the port a (here a port is a device or a file), which is associated with a security level (Slv_a, Gv_a). Then, PrcIFC will check the security of the output statement using the following rule

Rule 1. $(\cap_i Gp_{b_i} \cap Gp_a \neq \phi) \wedge (MAX(Slv_{b_i}) \leq Slv_a)$ (1)

The former part of the rule requires that the output port a and the variables b_i in the variable set should be compatible. Otherwise, the outputting is illegal. The latter part requires that the security level number of the output port should be at least the same as the variables in the variable set. This prevents leaking information carried by b_i to an output port. As to non-output statement, suppose a piece of information a, which is associated with a security level (Slv_a, Gv_a), is derived from the variable set "$\{b_i \mid b_i$ is a variable associated with the security level $(Slv_{b_i}, Gv_{b_i})\}$". Then, PrcIFC requires that the condition "$(\cap_i Gp_{b_i} \cap Gp_a \neq \phi)$" to be true, which means that the information a and variables b_i in the variable sets are compatible. If the condition is true, the following rule is used to adjust the security of the information a.

Rule 2. $Slv_a = MAX(Slv_{b_i})$ (2)

4. PROVE OF CORRECTNESS

Information leakage may occur during output, including outputting to files and devices.

Case 1. Outputting information to a device will not leak information under the control of PrcIFC.
Proof. Information output to a device *Dv* may only leak to persons. Suppose person *P* possesses a security level number SLV_p and $SLV_p < Slv_{Dv}$. If *Dv* and all information sources are in the same group, Rule 1 allows the output only when $MAX(Slv_{b_i}) \leq Slv_{Dv}$. If the output is allowed, *P* cannot access information output to *Dv* because $SLV_p < Slv_{Dv}$. Since b_i derives the information output to *Dv*, no b_i will be leaked to *P*.

Case 2. Outputting information to file will not leak information under the control of PrcIFC.

Proof. Information output to a file *Fi* may be leaked to persons or software. If *Fi* and all sources are in the same group, Rule 1 allows the output only when $MAX(Slv_{b_i}) \leq Slv_{Fi}$. Case 1 above states that no b_i will be leaked to a person *P* with a security level number SLV_p and $SLV_p < Slv_{Fi}$. On the other hand, if the software Sf_i intends to read the information of *V* from *Fi* to the variable V_1, Rule 2 causes Slv_v to be the same as Slv_{v_1}. If Sf_i outputs V_1 to a device, Case 1 states that V_1 will not be leaked to a person. If V_1 is output to another file, the proof goes back to the beginning of this proof. However, we know that the information of a sensitive variable may be: (a) operating by a software system, (b) output to a device, or (c) output to a file. Information operating by a software system will not be leaked without output. Information output to a device will not be leaked (see Case 1). As to the information output to a file, it will not be leaked to persons. #

5. EXPERIMENT RESULTS

We required students to develop a simple library system using UML as the development model. The target language is C. We organized two groups of students for the experiment, in which every group was consisted of ten students. For the first group, we required students to develop the system without PrcIFC. PrcIFC was embedded in their systems during implementation. That is, the students needed not use the prototyping language to write specifications and needed not use the target language to write design documents. On the other hand, we required students in the other group to follow the process in Figure 1 to develop their software. During software development, we required the students to collect the time they develop documents. During testing, we required students to collect the number of statements that violate PrcIFC. The collected data were then averaged. Table 1 depicts the averaged data, in which the development time of the first group is normalized to one.

Table 1. Experiment result

Group	Requirement analysis time	System design time	Implementation time	Statements violating PrcIFC
1	1	1	1	5.33
2	1.25	1.32	0.69	1.98

Table 1 shows that the requirement analysis and design time of the second group was more than that of the first group. This is reasonable because students in the group should write their specifications and design documents using the prototyping language and the target language, respectively. Moreover, PrcIFC should be embedded in the languages. On the other hand, the implementation time of the second group was much less because the design documents had been written in the target language. Moreover, students in the first group should embed PrcIFC in their systems. What we especially concerned is the statements that violate PrcIFC. The experiment result exercises us and we believe that PrcIFC is indeed valuable, because many statements that may leak information have been removed during software development phases before implementation.

6. CONCLUSION

Information flow control (IFC) is useful in preventing information leakage during software execution. According to our survey, we cannot identify an IFC model that is applied on the entire software development process. Applying IFC model on the entire software development process will not exclude the viewpoints of any stakeholders. Moreover, statements that may leak information can be identified during every phase of software development. The other problem of traditional IFC models is large runtime overhead. We cannot identify a model that can remove this overhead. According to the description above, we design a new IFC model named PrcIFC (process IFC). It offers the following major features:

1. PrcIFC is applied on the entire software development process. Therefore, it will not exclude the viewpoints of any stakeholders and helps correcting statements that may leak information during every phase of the development.
2. PrcIFC is disabled when software is online, which remove runtime overhead.

In the future, we will require students to apply PrcIFC on their exercises for checking the advantages and disadvantages of the model in depth. We hope to improve PrcIFC from the responses of students.

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”, to appear in 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] 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
[9] J. Bacon, D. Eyers, T. F. J. –M. Pasquier, J. Singh, and P. Piezuch, "Information Flow Control for Secure Cloud Computing", IEEE Trans. Network and Service Management, 11(1), pp. 76-89, 2014.
[10] S. -C. Chou, "Controlling Information Flows in Net Services with Low Runtime Overhead", International Journal of Computer Network and Information Security, vol. 7, no. 3, pp. 1-9, 2015.
[11] Myers A., and Liskov, B., 1998. Complete, Safe Information Flow with Decentralized Labels. Proc. 14'th IEEE Symp. Security and Privacy, 186-197.
[12] D. E., Denning, 1976. "A Lattice Model of Secure Information Flow", Comm. ACM, vol. 19, no. 5, 236-243, 1976.
[13] D. E., Denning, P. J., and Denning, Certification of Program for Secure Information Flow", Comm. ACM, vol. 20, no. 7, 504-513, 1977.
[14] Myers A., and Liskov, B., 2000. Protecting Privacy using the Decentralized Label Model. ACM Trans. Software Eng. Methodology, vol. 9, no. 4, 410-442.
[15] Samarati, P., Bertino, E., Ciampichetti, A., and Jajodia, S., 1997. Information Flow Control in Object-Oriented Systems. IEEE Trans. Knowledge Data Eng., vol. 9, no. 4, 524-538.
[16] Samarati, P., Bertino, E., Ciampichetti, A., and Jajodia, S., 1997. Information Flow Control in Object-Oriented Systems. IEEE Trans. Knowledge Data Eng., vol. 9, no. 4, 524-538.
[17] Yu, T., Winslett, M., Seamons, K., 2003. Supporting Structured Credentials and Sensitive Policies through Interoperable Strategies for Automated Trust Negotiation. ACM Transactions on Information and System Security, vol. 6, no. 1, 1-42.
[18] Koshutanski, H., Massacci, F., 2005. Interactive Credential Negotiation for Stateful Business Processes, Lecture notes in computer science, 256-272.
[19] Bhatti, R., Bertino, E., Ghafoor, A., 2004. A Trust-based Context-aware Access Control Model for Web Services. IEEE ICW'04, 184 – 191.
[20] Bhatti, R., Ghafoor, A., Bertino, E., Joshi, J. B. D., 2005. X-GTRBAC: An XML-Based Policy Specification Framework and Architecture for Enterprise-Wide Access Control. ACM Transactions on Information and System Security (TISSEC), Vol. 8, No. 2, 187 – 227.
[21] Bertino, E., Bonatti P. A., Ferrari, E., 2001. TRBAC: A Temporal Role-Based Access Control Model. ACM Transactions on Information and System Security, Vol. 4, No. 3, 191 – 233.
[22] Wonohoesodo R., Tari, Z., 2004. A Role Based Access Control for Web Services. Proceedings of the 2004 IEEE International Conference on Service Computing, 49-56.
[23] Sandhu, R. S., Coyne, E. J., Feinstein, H. L., Youman, C. E., 1996. Role-Based Access Control Models. IEEE Computer, vol. 29, no. 2, 38-47.
[24] R. Sandhu, D. F., Ferraiolo, and D. R., Kuhn, D. "The NIST Model for Role Based Access Control: Toward a Unified Standard", 5th ACM Workshop Role-Based Access Control. pp. 47–63, 2000
[25] W. She, I. -L. Yen, B. ThuraiSingham, and S. –Y. Huang, "Rule-Based Run-Time Information Flow Control in Service Cloud", 2011 IEEE International Conferences on Web Services, pp. 524-531, 2011.
[26] L. Sfaxi, T. Abdellatif, R. Robbana, and Y. Lakhnech, " Information Flow Control of Component-Based Distributed Systems", Concurrency and Computation: Practice and Experience, available on http://www.bbhedia.org/robbana/Wiley.pdf
[27] JIF website, "Jif: Java + information flow", available on http://www.cs.cornell.edu/jif/
[28] 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.
[29] S. -C. Chou, "A Prototyping Technique to Verify Requirements", ICS 2002, Hualien, Taiwan.

AUTHORS

Shih-Chien Chou is currently a professor in the Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan. His research interests include software engineering, process environment, software reuse, and information flow control.