# Dynamically Preventing Information Leakage for Web Services using Lattice

Shih-Chien Chou
Department of Computer Science and Information Engineering
National Dong Hwa University, Taiwan
E-Mail: scchou@mail.ndhu.edu.tw

***Abstract-* The use of web services becomes important. Since web services may be provided by unknown providers, many researchers designed mechanisms to ensure secure access of web services. Existing mechanisms generally statically decides whether a requester can invoke a web service but ignore the importance of dynamically preventing information leakage during web service execution. This paper proposes a lattice-based information flow control model WSIFC (web service information flow control) to dynamically prevent information leakage. It uses simple rules to monitor the flows of sensitive information during web service execution.**

***Keyword- Web service, information flow control, information leakage prevention***

## I. INTRODUCTION

Many researchers design mechanisms to ensure secure access of web services. Existing researches generally *statically* decide whether a requester can invoke a web service using mechanisms such as attributes and credentials. In our opinion, the static operation is insufficient. It is also necessary to *dynamically* ensure web service security. Dynamically ensuring web service security prevents information leakage *during web service execution*. This prevention is important because a web service may be dishonest or malicious, which may leak sensitive information received from requesters. Even an honest and friendly web service may also incidentally leak information during web service execution. For example, suppose a requester sends his credit card information to an honest web service for a payment. If the web service incidentally displays the credit card information on the screen and the information is captured by a malicious person, information leakage occurs.

According to the above description, dynamically preventing information leakage during web service execution is crucial. The prevention can be achieved by an *information flow control* model [1-2], which ensures that every information flow is secure during software execution. An *information flow* may occur under the following situation: (a) assign an expression result to a variable, (b) read an input device, (c) write an output device, and (d) send information to another site. Our previous work proposed an information flow control model for web services [3], which is composed of a *trust management mechanism* and an information flow control model. The former mechanism statically checks whether a requester can invoke a web service and the latter model dynamically prevents information leakage during web service execution. Although the dynamic information flow control model prevents all information leakage, its runtime overhead is large. To reduce the overhead, we use a far simpler approach to design a new information flow control model WSIFC (web service information flow control) for web services. WSIFC is a pure dynamic model. It reuses the trust management mechanism of our previous work to statically decide whether a requester can invoke a web service.

## II. RELATED WORK

XACML [4] is a standard offering mechanisms to describe access policies for web services. The model proposed in [5] is an implementation of XACML. It allows requesters to use PMI (privilege management infrastructure) for managing the checking, retrieval, and revocation of authentication. The model also uses an RBAC-based web service access control policy to determine whether a requester can invoke a web service. The model proposed in [6] uses X-GTRBAC [7] to control the access of web services. X-GTRBAC can be used in heterogeneous and distributed sites. Moreover, it applies TRBAC [8] to control the factor related to time. The model in [9] determines whether a composition of web services can be invoked. It offers a language to describe policies, which check whether a composition of web services can be invoked. The model in [10] uses RBAC (role-based access control) concept [11] to define policies of accessing a web service. It is a two-leveled mechanism. The first level checks the roles assigned to requesters and web services. An authorized requester is a candidate to invoke a web service. The second level uses parameters as service 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.

In addition to the researches discussed above, quite a few researches focus on negotiation among requesters and web services. In general, negotiation can be regarded as access control. We survey some of them. The model Trust-Serv [12] uses state machines to dynamically

Sponsor of this work: NSC-99-2221-E-259-008, Taiwan

choose web services at run time. The choosing is a kind of negotiation. It uses trust negotiation [13] to select web services that can be invoked. The kernel component to determine whether a web service can be invoked is *credential*. The model in [14] uses digital credentials for negotiation. It defines strategies for negotiation policies. The model in [15] handles k-leveled invocation of web services. The primary component used in the model is credentials.

According to our survey, existing models generally statically determine whether a requester can invoke a web service. As to dynamically preventing information leakage during web service execution, they do nothing.

## III. THE MODEL

Our previous work uses two time-consuming rules and a complicated *join* operator to control *every variable access* [3]. This results in large runtime overhead. In our opinion, sending sensitive information to a web services may be unavoidable. However, requesters may not send too much sensitive information to web services according to security consideration. For the web services whose providers are not trusted by a requester, the requester may even send no sensitive information to the web services. To reduce runtime overhead, our model WSIFC controls only the access of sensitive variables.

WSIFC is based on lattice. A lattice is a *partial ordered* graph consisting of a min-node and a max-node. If a lattice is used to control information flows, the min-node possesses the least privileges while the max-node possesses the most ones. The traditional lattice-based model uses the "no read up" and "no write down" rules to control information flows (note that the "can flow" relationships in [1] inherits the spirit of the two rules). If the security levels of sensitive information are structured using the lattice model and the two rules are obeyed, no information leakage will occur. However, the traditional lattice-based information flow control model is a MAC (mandatory access control), which is criticized as too restricted. To loosen the restriction, WSIFC only places sensitive variables in a lattice and uses simple rules to replace the restricted "no read up" and "no write down" rules.

Before formally defining WSIFC, we first describe information leakage in more details. Generally, *information leakage occurs when malicious persons or web services illegally obtain others' sensitive information*. In general, persons can obtain information from screen or files, and web services can obtain information from files or other web services (either executing on the same site or on the other ones). Information leakage may thus happen when: (a) sensitive information is output to screens or files or (b) sensitive information is transferred to other web services. In our research, we suppose information transferring on the network is secure and thus skip the problem of cryptography. We also skip the effect of viruses and worms because they are out of the scope of our research. Below we turn our focus back to WSIFC.

WSIFC uses *security levels* to decide whether an instruction may leak sensitive information during the execution of a web service. It attached a security level to every sensitive variable. Since a lattice is partial ordered and WSIFC is lattice-based, security levels should also be partial ordered. To achieve this, a security level in WSIFC is defined as: (*Gp, security level number*), in which *Gp* is a *group number* and *security level number* decides the sensitivity of the variable. To compare or exchange information associated with security levels, the information should be within the same group, which achieves partial ordering. Partial ordering is necessary when variables are incomparable. For example, a variable storing an American's salary with a unit of USD cannot be compared with one storing a Taiwanese's salary with a unit of NTD.

According to the above description, the group number in a security level achieves partial ordering and the security level number decides the sensitivity of a variable. A security level number is between *0* and *n*, in which *0* means the less sensitive information and *n* the most sensitive one. The number *n* is decided by the user, in which larger *n* results in finer-grained control. In addition to security level, a sensitive variable is also associated with a tag, which is a set. A component in the set is an IP address plus a port number pair (port numbers represent web services in the IP). Tag decides whether sending information of a sensitive variable to another web service is secure. If the IP address plus port number pair of a web service is within a sensitive variable's tag, information of the variable can be sent to the web service.

By now we have discussed sensitive variables. As to non-sensitive ones, they possess neither security levels nor tags. That is, non-sensitive variables are not placed in a lattice. Nevertheless, the security level and tag of a variable may be adjusted during web service execution, which implies that a non-sensitive variable may become sensitive and vice versa during runtime. For example, after executing the statement "a=b;", the security level and tag of variable *a* is adjusted to be the same as those of *b*. If *b* is sensitive, *a* becomes sensitive.

As mentioned before, sensitive information managed by a web service may be leaked only when: (a) it is output to files or screens or (b) when it is transferred to other web services. The latter case of leakage can be prevented by tags associated with sensitive variables. To achieve the former leakage prevention, every file and screen that may receive output information should also be associated with a security level. Generally, the security level of a screen is decided by its location. The only possibility of changing a screen's security level is changing its location. However, changing screen location is only known by persons and web service will never know that. Therefore, the security levels of screens will be unchanged during web service execution. As to files, their security levels also cannot be adjusted because the adjustment will affect the access of the existing information within the files. For example, if the security level number of a file is 3, it can be accessed by a user possessing a privilege to access files whose security level number is 3 or lower. If the security level of a file is adjusted to be 4, the user mentioned above can no longer access the file. Adjusting the security levels of files is thus infeasible. In addition to writing a file, reading a file may also cause information leakage. For example, if a low sensitive variable intends to read the contents of a high sensitive file, rules should be defined to prevent information leakage. We will give information leakage prevention rules for WSIFC after defining the model.

The kernel concepts of WSIFC have been described in details above. Below we give a definition to WSIFC. To prevent information leakage during the execution of a web service, WSIFC should be embedded in the web service.

**Definition 1**. *WSIFC* = (*SV*, *SC*, *SF*, *SVSL*, *SCSL*, *SFSL*, *SVTAG*, *MIN*, *MAX*), in which

a. *SV* is the set of sensitive variables. Initially, every variable in a web service is non-sensitive. When the web service is invoked and receives sensitive information sent from a requester, the parameters receiving sensitive information become sensitive. During web service execution, sensitive parameters may be assigned to other variables, which results in more sensitive variables that should be monitored.
b. *SC* is the set of sensitive screens. The security levels of screens are decided by their locations and will be unchanged during web service execution.
c. *SF* is the set of sensitive files. The security levels of sensitive files cannot be changed.
d. *SVSL* is the set of security levels for sensitive variables. A security level is composed of a group number and a security level number.
e. *SCSL* is the set of security levels for sensitive screens. The security level of a sensitive screen cannot be changed during the execution of a web service.
f. *SFSL* is the set of security levels for sensitive files. The security level of a sensitive file cannot be changed during the execution of a web service.
g. *SVTAG* is a set of tags. If the information of a sensitive variable can be transferred to other web services, the variable should be associated with a tag containing a set of IP address plus port number pairs.
h. *MIN* is the min-node of WSIFC.
i. *MAX* is the max-node of WSIFC.

For simplification purposes, we do not include other I/O devices such as keyboards and mice in Definition 1. WSIFC uses the same approaches to control information flows to/from I/O devices other than files and screens. If a device can be read, its control is the same as reading a file. If information can be output to a device and the information can only be accessed by persons, the control is the same as writing information to a screen. If information can be output to a device and the information can be accessed by persons and web services, the control is the same as writing a file.

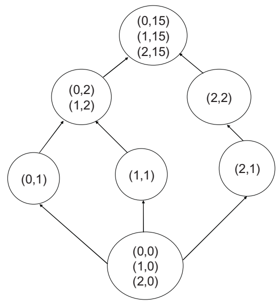

Figure 1. An example lattice of WSIFC

In the lattice-based WSIFC, a node may contain one or more security levels. Moreover, since variables, files, screens, and other I/O devices may be in the same security level, a security level in WSIFC may be associated with multiple sensitive items such as variables and files. Figure 1 depicts an example lattice of WSIFC, which only shows the security levels but not the sensitive items associated with the security levels. In the following paragraphs we will define the rules to control information flows using WSIFC.

**Rule 1**. Sensitive variables appear in a *derivation statement* should be in the same group. Here a derivation statement is *one that will cause information flows such as assignment statement*. As to non-sensitive variables or constants, they can appear in any statement and will not affect the execution of the statement. Rule 1 is an umbrella rule constraining the others. That is, if Rule 1 is violated, no derivation statement can be executed.

**Rule 2**. If the variable *d_var* is derived from variables in the sensitive variable set "{$var_i$| $var_i$ is a sensitive variable and *i* is between *1* and *n*}" and other non-sensitive variables, WSIFC allows *d_var* to receive the derived information in any case. After the assignment, the security level number and tag of *d_var* are respectively changed to $MAX(SLV((\text{var}_i)_{i=1}^{n}))$ and $\bigcap_{i=1}^{n} tag_i$, in which:

a. The function *SLV* extracts a sensitive variable's security level number from its security level.
b. The function *MAX* extracts from a set of numbers the maximum one.

c. $tag_i$ is the tag of the sensitive variable $var_i$.

Rule 2 may confuse many readers because it completely violates the "no read up" and "no write down" rules of the traditional lattice-based model. In addition, it controls neither read nor write access. We explain Rule 2 as follows. Before that, we make a crucial assumption, which is: *every data assigned to a variable is reliable*.

Existing information flow control models generally do not allow information with high security level to be assigned to low security level variable. On the other hand, WSIFC allows every assignment. To prevent leakage when sensitive information is output, WSIFC changes the security level and tag of a variable receiving a value as mentioned above. Suppose variable *d_var* is assigned a value derived from a sensitive variable set $\{var_i\}$. Then, the security level number of *d_var* will be changed to $MAX(SLV((\mathrm{var}_i)_{i=1}^{n}))$. According to the change, the security level of *d_var* will be upgraded if the initial *SLV*(*d_var*) is smaller than $MAX(SLV((\mathrm{var}_i)_{i=1}^{n}))$. The upgrading prevents the variable *d_var* from being leaked. On the other hand, if the initial *SLV*(*d_var*) is larger than $MAX(SLV((\mathrm{var}_i)_{i=1}^{n}))$, the security level of *d_var* will be downgraded. Under this situation, the information obtained by *d_var* will also not be leaked. According to the crucial assumption "*every data assigned to a variable is reliable*" mentioned above, both read and write access are secure according to Rule 2.

Below we use the statement "a=b+c+d;" to explain Rule 2 (here we bypass the discussion of tags, which will be discussed in Rule 6). Suppose the initial security levels of the variables *a*, *b*, *c*, *d*, are respectively (1,1), (1,2), (1,3), and (1,4). In this case, the security level of the variable *a* is the smallest and existing information flow control model will ban the assignment. However, WSIFC allows the above statement to execute. After the execution, WSIFC changes the security of variable *a* to be (1,4). According to the change, the security level of variable *a* have been upgraded and therefore no information leakage will occur. On the other hand, if the initial security level of variable *a* is (1,5), the security level of variable *a* will be set (1,4) after the assignment.

Rule 2 will become non-secure only when the above crucial assumption isn't fulfilled. In this case, low security level variables may maliciously corrupt the information carried by high security level variables (note that even in this case, no information leakage will occur). We cannot estimate the possibility of the corruption. However, as the assumption is a compromise to reduce runtime overhead, taking risk is unavoidable. We think that making a compromise to bypass write check in Rule 2 is acceptable because the compromise may only corrupt but not leak information. Since the primary objective of WSIFC is preventing information leakage, a compromise that will not hurt the primary objective is not that serious.

**Rule 3**. To output the information of a sensitive variable *SVAR* to a screen *SCRN*, *SLV(SCRN)* should be at least as large as *SLV(SVAR)*. Remember that the function *SLV* extracts a sensitive variable's security level number. This rule avoids high sensitive information to be captured by persons that can only read low sensitive information from a screen.

**Rule 4**. If a variable *VAR* intends to read information from a file *FLE*, the reading operation is allowed. After the reading, the security level and tag of *VAR* should be set the same as those of the information read from *FLE*. The spirit of this rule is the same as Rule 2, because *reading a value from a file* corresponds to *deriving a value from the file and then assigning the value to the variable*.

**Rule 5**. If a sensitive variable *VAR* intends to write its information to a file *FLE*, *SLV(FLE)* should be at least as large as *SLV(VAR)* . Reading and writing a file are totally different. When reading information from a file to a variable, the information being read can be protected as long as the security level and tag of the variable are set the same as the information. On the other hand, since the security level of a file cannot be changed as mentioned before, writing information with high security level to a low security level file may result in information leakage.

**Rule 6**. To send the information of a sensitive variable to another web service, the service's IP address plus port number pair should be within the tag of the variable. In the sending operation, the information and the security level of the variable should be sent together so that the web service receiving the information can protect the information using WSIFC.

## IV. PROOF OF CORRECTNESS

Information leakage may occur under the following cases: (a) the information of a variable is output to a screen, (b) the information of a variable is output to a file, and (c) the information of a variable is sent to web services in other sites. The above cases reveal that information leakage may occur only when variable contents are output (sending information to other sites can be considered a special case of output). Below we prove that none of the above mentioned cases will leak information.

Case 1: *Outputting the information of a sensitive variable to a screen will not leak information*. To prove this case, we make the following assumptions: (a) *SLV(VAR)* of the sensitive variable *VAR* is *m* and (b) *VAR* is output to a screen *SCRN* whose *SLV(SCRN)* is *n*. If a person intends to access the screen *SCRN*, the person should possess a privilege to access screens whose security level numbers are at least *n*. Since $n \geq m$ according to Rule 3, Case 1 is true.

Case 2: *Outputting the information of a sensitive variable to a file will not leak information*. Information output to a screen may only be leaked to persons. However, information output to a file may be leaked to persons or leaked by web services that read the file. Both the above two situations should be considered for Case 2. Proving information output to a file will not be leaked to persons is similar to that in Case 1. We do not duplicate the proof. Below we prove information output to a file will not be leaked by web services that read the file. Here we only consider web services that will not transfer information to other sites because the type of web services will be discussed in Case 3 below.

To prove Case 2, we assume that: (a) *SLV(VAR)* of the sensitive variable *VAR* is *m*, (b) *VAR* is output to a file *FLE* whose *SLV(FLE)* is *n*, and (c) WSIFC is embedded in every web service that will access *FLE*. The third assumption is crucial because WSIFC cannot control information flows of a web service not embedding WSIFC. According to Rule 5, $n \geq m$. If a web service reads *VAR* from *FLE* and assigns its information to the variable *VAR1*, *SLV(VAR1)* should be set the value *m* according to Rule 4. If *VAR1* is output to a screen, the proof of Case 1 ensures that the output information will not be leaked to persons. If *VAR1* is output to a file *FLE1*, the proof goes back to the beginning of this proof. That is, this proof is endless. Although the proof is endless, we know that the information of a sensitive variable may be: (a) operated by a web service, (b) output to a screen, or (c) written to a file. The information operated by a web service will not be leaked because no output occurs. The information output to a screen will not be leaked as described above. As to the information output to a file, Rules 3 through 5 ensures that the information will not be leaked to persons. Since no information will be transferred to other sites in this case, no information will be leaked to persons corresponds to no information leakage. According to the above description, Case 2 is true.

Case 3: *Sending the information of a sensitive variable to web services in other sites will not leak information*. Suppose the web service *WEBSER* executing on another site receives the information of a sensitive variable *VAR*. To ensure *WEBSER* will not leak the information it receives, WSIFC should be embedded in *WEBSER*. As Rule 6 required, *WEBSER* is within the tag of *VAR*, which means that *WEBSER* is trusted by *VAR*. Suppose the parameter *PAR* in *WEBSER* receives the information of *VAR*. Since receiving the information of an argument by a parameter can be regarded as an assignment statement, the security level and tag of *PAR* will be set the same as those of *VAR* according to Rule 2. After the argument has been received, WSIFC will control the information flows of *WEBSER* during web service execution. According to Cases 1 and 2 above, no information leakage will occur in a normally executing web service that embeds WSIFC. Therefore, Case 3 is true.

## V. PROBLEM DISCUSSION

Many problems should be solved before WSIFC can be widely applied. We discuss some as follows.

1. *WSIFC should be embedded in every web service*. To use WSIFC for information leakage prevention during web service execution, WSIFC should be embedded in every web service. It is really a problem to require every service provider to embed WSIFC in the web services they provided. Perhaps someday every requester will understand the importance of preventing information leakage during web service execution. At that day, a standard model for the prevention will be defined. When the day comes, we hope that more or less of the concepts in WSIFC can be included in the standard.
2. *The definition of security levels between requesters and that within the web services should be the same*. We use an example to explain this. Suppose a requester thinks that a variable with a security level number *8* is only accessible by managers or employees with higher grades but a web service thinks this security level is accessible by vice managers or employees with higher grades. With the assumptions, the information sent to the web service may be leaked to vice managers. To solve this problem, a standard seems necessary.
3. *Even the above problems have been solved, information leakage may still occur*. This problem may happen in a web service provided by dishonest or malicious service providers. To solve this problem, we suggest: (a) don't send highly sensitive information to web services provided by unfamiliar providers, (b) invoke web services provided by the providers trusted by a requester whenever possible, and (c) never invoke any web service of a provider if it used to leak information. The third solution may be the most important because it is a punishment. This problem reveals that ensuring the security of

every service invocation is difficult. Perhaps this is the most serious problem to solve because requiring every service provider to be honest is almost impossible.

There are still other problems. We do not intend to discuss more because many problems are not within the technical area but within the human management area. We think that the problems are out of the scope of our research.

## VI. CONCLUSION

Many researchers designed mechanisms to ensure secure access of web services. According to our survey, existing models generally *statically* decide whether a requester can invoke a web service but ignore the importance of *dynamically* preventing information leakage during web service execution. This paper proposes an information flow control model WSIFC (web service information flow control) for the prevention. WSIFC is based on lattice. The traditional lattice-based information flow control model is a MAC (mandatory access control) and is criticized as too restricted. To overcome this, WSIFC uses simple rules to replace the "no read up" and "no write down" rules for MAC. To reduce runtime overhead, WSIFC only monitors the flows of sensitive information using simple rules.

WSIFC should be embedded in every web service. If a web service does not embed the model, information leakage may occur when the web service is invoked. Since web services are provided by providers around the world, we cannot require every provider to embed WSIFC in their web services. This is a problem almost without solution. Perhaps the software world needs a standard for embedding an information flow control model in every web service. If the standard is finally defined, we hope that the important concepts proposed by WSIFC can be used.

## REFERENCES

[1] Denning, D. E., 1976. A Lattice Model of Secure Information Flow. *Comm. ACM*, vol. 19, no. 5, 236-243.

[2] Chou, S. –C., 2004. Embedding Role-Based Access Control Model in Object-Oriented Systems to Protect Privacy. *Journal of Systems and Software*, 71(1-2), 143-161

[3] Chou S. –C., and Huang, C. –H., 2010. An Extended XACML Model to Ensure Secure Information Access for Web Services. *Journal of Systems and Software*, vol. 83, no. 1., 77-84.

[4] OASIS, 2003. eXtensible Access Control Markup Language (XACML) Version 1.0. *OASIS Standard 18*.

[5] Shen, H. -B., Hong, F., 2006. An Attribute-Based Access Control Model for Web Services. *IEEE International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT'06)*, 74-79.

[6] Bhatti, R., Bertino, E., Ghafoor, A., 2004. A Trust-based Context-aware Access Control Model for Web Services. *IEEE ICW'04*, 184 – 191.

[7] 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*, Vol. 8, No. 2, 187 – 227.

[8] 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.

[9] Seamons, K. E., Winslett, M., Yu, T., 2001. Limiting the Disclosure Access Control Policies during Automated Trust Negotiation. *Network and Distributed System Security Symposium.*

[10] 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.

[11] 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.

[12] Skogsrud, H., Benatallah, B., Casati, F., 2004. Trust-Serv: Model-Rriven Lifecycle Management of Trust Negotiation Policies for Web Services. *International World Wide Web Conference*, 53-62.

[13] 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.

[14] Koshutanski, H., Massacci, F., 2005. Interactive Credential Negotiation for Stateful Business Processes, *Lecture notes in computer science*, 256-272.

[15] Mecella, M., Ouzzani, M., Paci, F., Bertino, E., 2006. An Access Control Enforcement for Conversation-based Web Services. *International World Wide Web Conference*, 257-266.