Policy Description Language for Dynamic Access Control Models

Masaki HASHIMOTO†, Mira KIM†, Hidenori TSUJI† and Hidehiko TANAKA†
†*Graduate School of Information Security, INSTITUTE of INFORMATION SECURITY*
2-14-1 Tsuruya-cho, Kanagawa-ku, Yokohama, 221-0835, Japan
Email: {dgs074105, mira, hide, tanaka}@iisec.ac.jp

***Abstract*—Recently, dynamic access control models are proposed to restrict access domain appropriately in Multi-Layered Defense. However, policy description languages proposed so far can not express the models effectively in proper granularity. In this paper, we propose a policy description language which can designate precise condition for access control by using dynamic status of application process. Using the proposed language, we compose the policy of SELinux which is major implementation achieving Multi-Layered Defense and confirm the advantages of the proposed language by evaluating the response and the expressiveness.**

***Keywords*-Access control; Computer security;**

I. Introduction

In this paper, we propose a policy description language[1] which can describe the dynamic access control models to expand the application range of Multi-Layered Defense. Our language expresses the dynamic status of processes as the ground for authorizations, though languages proposed so far describe access control rules based on the static information like subjects, objects and operations.

Our language verifies the information flow of the future or the past by reasoning, and uses the results as the access control information for authorization. We present the implementation of the proposed language with Datalog[2], the subset of Prolog[3], and explain about the concrete way of describing the access control rules based on the future/past information flow and state designation of the processes.

As an experiment which demonstrates the validity of our language, we compose the policy of SELinux and evaluate the response performance and the equality of the contents of responses between original SELinux and our language.

II. Related Works and Motivation

In this chapter, we show the related works and explain the motivation of our research. First, we show the policy of SELinux as an actual example of policy description problems, then recent researches which focus policy descriptions are presented. Finally, we explain the purpose of our research.

A. Related Works

In this section, we show the outline of SELinux as a preceding research which achieves Multi-Layered Defense first, and explain the problems related to the policy descriptions. After that, several researches about the policy description language in recent years are presented.

1) Policy Description Problems in SELinux: SELinux is the implementation of the Flask security architecture[4] for Linux kernel, mainly composed by the security server, the object manager and the access vector cache. SELinux evaluate the access control policy and forces access control decisions by hooking system call functions which relate to the access control in the kernel space. Because the authorization can be evaluated with system call functions, it is possible with SELinux that models like RBAC and DTE are achieved in the granularity of the system call level. However, describing fine-grained access control rules is highly difficult for the human being on the other side, though they must be described to control the accesses in the fine-grained granularity.

For instance, SELinux's policy is composed by lining up access control rules as the followings.

allow $acct\_t\ unconfined\_t{:}fd\ use$;
allow $acct\_t\ init\_t{:}process\ sigchld$;
allow $acct\_t\ init\_t{:}process\ signull$;
allow $acct\_t\ rpm\_t{:}fd\ use$;

In the SELinux's policy, one authorization rule is expressed in each line. The first item "allow" in each line shows the line as a rule to permit some action. The second item designates the information of the subject which this rule is applied to, and the information of the object and operation are designated for the rest. The rules by the syntax are enumerated over the 40,000 lines in the simplest edition policy for the individual users in the SELinux.

As a result, even if each rule can be understood intuitively, it is difficult to predict what occurs actually with the policy constructed as the collection of those lines. Additionally, the situation must be more difficult in the near future, because the techniques have been already proposed which expand the application range of the mandatory access control mechanism to the application layers and other networked systems from the operating system layer of a single system.

2) Recent Researches: SDSI[5] and SPKI[6] are the framework to evaluate reliability between the system by exchanging a certificate. SDSI uses the predicate of "speaks-for" to building the logic architecture and SPKI can encode the predicates in the declaration as a tag. Furthermore, they has the characteristics to express a policy as an English writing, but SDSI can not define and use the corresponding predicate to the applications because it can use only speaks-for as a predicate and SPKI can not define the suitable predicate to the applications easily because of the restriction from the tags though it uses more than one predicate. Though the various conditions which are necessary for the authorization decision can be defined, as for PolicyMaker[7] and KeyNote[8] as well, the subject which can not define the optional predicate corresponding to the application is reported in the same way. SecPAL[9] is the policy description language by the constraint logic programming and demands for access are authorized officially when query to the set of the clauses succeeds. The syntax of SecPAL is close to the natural language, and semantics is composed by three deduction rules. And it can express many access control models by supporting various expressions like the delegation of authority, the negation and the recursion, but it focuses on the control of more than one management domain by the easy and clear policy description technique, hence, the examination for the dynamic elements is not the main target.

B. *Motivation of Our Research*

In this paper, we propose the expressive policy description language which supports dynamic Multi-Layered Defense with fine-grained granularity by using the process status information and the results of information flow verification as the ground of authorizations. Our language defines layer composition dynamically by evaluating values to change during the system operations in addition to them, and aim to express the access control models of fine-grained granularity beyond before. As for the description of the access control models based on the dynamic information, recent researches are not enough because they focuses in the technique to express delegation of authority briefly between the several domains which are managed by own policies. Paying attention to this point, our proposed language gives the authorization decision to mandatory access control mechanism based on the process status informations and the future and/or past information flow, supporting access control models which consider the dynamic conditions.

For instance, it can be expected to give the least authority set to the agent process for the other system corresponding to the status of tasks, or to exclude the future rivalry of authority set between tasks in advance, and so on. And, by verifying past information flow, the authorization decision can be changed flexibly whether the process pass through the trusted authentication mechanism which is specified by policy. Furthermore, it becomes possible to prevent the information flow which does not be intended by controlling authorizations based on the dynamic status of the specific process and the system, such as ”Disabled” and ”Polluted”. Additionally, it can be expected to reduce the difficulty of the policy description shown with the example of SELinux as a result of the improvement in expression, because more access control rules can be expressed in less descriptions.

III. Policy Description Language

In this chapter, we present the basic design of the proposed language. First, we define the syntax and semantics for the declarative statement and authorization query of the policy. Next, we show several deductive rules to apply to them. The proposed language adopts the syntax and semantics to be similar to English which is intuitively easy to understand, and deductive rules are very simple. The predicate logic is the foundation of the our language.

A. *Syntax and Semantics*

We compose a policy as a set of the declarative statement (DS: Declarative Statements) and access control decisions are decided by Q (authorization queries) described in the following syntax. Accesses are permitted or denied by that answer.

$$\begin{aligned}
DS &::= \textbf{Policy Specifies } fact \textbf{ if } fact_1,..., fact_n \\
Q &::= \textbf{Policy Specifies } fact \mid \exists x(Q) \\
E &::= x \mid A \\
T &::= claim\ E_1 \textbf{ for } E_2 \textbf{ in } E_3 \\
V &::= \textbf{is authorized to } T \mid \textbf{will be authorized to } T \\
&\quad \mid \textbf{has role } E \mid \textbf{has type } E \\
&\quad \mid \textbf{has state } E \mid \textbf{role trans } E \\
fact &::= E\ V \\
claim &::= \textbf{allow} \mid \textbf{auditallow} \mid \textbf{dontaudit} \\
&\quad \mid \textbf{neverallow} \mid \textbf{transition}
\end{aligned}$$

Bold forms represent fixed numbers, predicates and variables which do not influence the semantics. **Policy** shows the subject which defines each DS, and **Policy Specified** is always fixed in this syntax. $facts$ shows the actual contents of the access control rule and $fact_1$,... $fact_n$ are conditional facts. As for Q, the conditinal clause of DS was removed and Q express the confirmation of the existence of the DS and constants are substituted for variables as passible.

E denote constants and variables. Constants means subjects or objects for authorization and they represent processes, domains, roles, and so on. We designate them as alphabetical one character in the capital letter of italics fonts. Variables are restricted its domain within the domain of constants. Therefore, $fact$s and DS are not substituted for variables. T denote objects and operation for access in the DTE model; E_1 for the contents of the operation, E_2 for the object classes and E_3 for the types of the object. And $claim$ designates the kind of authorization. **has role**, **has type** and **has state** express the attributes which the subject holds respectively. **role trans** designates whether the subject could transit to the role designated with objects. $claim$ is equivalent to the predicate of the $fact$ and they designate the kind of permission.

We define the semantics of Q by the relations designated by $P, \theta \vdash Q$ in the following. P denotes the policy, a set of DS. θ is the mapping function which maps the variables to the constants or to the other variables and *vars(X)* designates the variable clause which exists in the clause X. We interpret Q as access control decision by the truth evaluation.

$P, \theta \vdash$ **Policy Specifies** *fact*
iff $P, \theta \models$ **Policy Specifies** *factθ,*
and $dom(\theta) \subseteq$ *vars(***Policy Specifies** *fact)*
$P, \theta\_x \vdash \exists x(Q)$ *iff* $P, \theta \vdash Q$

B. Deductive Rules

In the proposed language, it is possible to reduce the amount of description of policy by using the deductive rules and the formal verification of information flow expressed in the rest becomes possible.

RULE (a)

$$\frac{\begin{array}{c}(\textbf{Policy Specifies}\ \textit{fact}\ \textbf{if}\ fact_1, ...,\ fact_k) \in P \\ P \models \textbf{Policy Specifies}\ fact_i\theta\ \textit{for all}\ i \in \{\textit{1..k}\} \\ \textit{vars(fact}\theta\textit{)} \doteq \dot{O}\end{array}}{P \models \textbf{Policy Specifies}\ \textit{fact}\theta}$$

RULE (b)

$$\frac{\begin{array}{c}P \models \textbf{Policy Specifies}\ A\ \textbf{has type}\ B \\ P \models \textbf{Policy Specifies}\ B\ \textbf{will be authorized to}\ C\end{array}}{P \models \textbf{Policy Specifies}\ A\ \textbf{will be authorized to}\ C}$$

RULE (c)

$$\frac{\begin{array}{c}P \models \textbf{Policy Specifies}\ A\ \textbf{has role}\ B \\ P \models \textbf{Policy Specifies}\ B\ \textbf{has type}\ C\end{array}}{P \models \textbf{Policy Specifies}\ A\ \textbf{has type}\ C}$$

RULE (d)

$$\frac{\begin{array}{c}P \models \textbf{Policy Specifies}\ A\ \textbf{has role}\ B \\ P \models \textbf{Policy Specifies}\ B\ \textbf{role trans}\ C\end{array}}{P \models \textbf{Policy Specifies}\ A\ \textbf{has role}\ C}$$

RULE (e)

$$\frac{\begin{array}{c}P \models \textbf{Policy Specifies}\ A\ \textbf{has type}\ B \\ P \models \textbf{Policy Specifies}\ B\ \textbf{will be authorized} \\ \textbf{to transition}\ C\ \textbf{for}\ D\ \textbf{in}\ E\end{array}}{P \models \textbf{Policy Specifies}\ A\ \textbf{has type}\ C}$$

RULE (a) deduces the statement with no variables from the statement with conditional clauses and θ which makes each *fact* in the conditional clauses *vars(factθ)*$\doteq \dot{O}$. RULE (b) deduces the statement which designates A will be authorized to C from the statement expressing A has type B and the statement expressing B will be authorized to C. RULE (c) deduces the statement which designates Type C is assigned to A from the statement expressing Role B is assigned to A and the statement expressing Type C is assigned to Role B. RULE (d) deduces the statement which designates Role C is assigned to A from the statement expressing Role B is assigned to A and the statement expressing Role B transit to Role C. RULE (e) deduces the statement which designates Type C is assigned to A from the statement expressing Type B is assigned to A and the statement expressing Type C is assigned to the target objects when Type B generates Class D in Type E.

IV. IMPLEMENTATION

In this chapter, we present the implementation of the proposed language using Datalog and explain actual descriptions of the access control models based on the dynamic elements. We treat the authority set which will be able to be held in the future or the past as the information flow.

A. Implementation of Proposed Language in Datalog

We implement the description language by using Datalog, the subset of Prolog. Each declarative statement by the proposed description technique is translated as a clause of the Datalog program, and authorization decision sentence is evaluated by tabled resolution as an query for the program.

First, we defines the term of Datalog by followings. Literal L has the name of the predicate and a series of arguments. An argument takes constant or variable. A clause is written as $L_0 \leftarrow L_1, ..., L_n$ and composed by one head literal in the left part of "$\leftarrow$" and the lists of the body literals in the right-hand side. Each predicate of the proposed description technique translated into the Datalog literal as follows. The left part of "$\rightarrow$" is the description by the proposed language and right-hand side is a Datalog literal after the translation.

Policy Specifies E_1 **is authorized to claim** E_2 **for** E_3 **in** E_4
$\rightarrow$ *isAuthedto(claim,* E_1*,* E_2*,* E_3*,* E_4*)*
Policy Specifies E_1 **will be authorized to claim** E_2 **for** E_3 **in** E_4
$\rightarrow$ *willbeAuthedto(claim,* E_1*,* E_2*,* E_3*,* E_4*)*
Policy Specifies E_1 **has role** E_2 $\rightarrow$ *hasRole(*E_1*,* E_2*)*
Policy Specifies E_1 **has type** E_2 $\rightarrow$ *hasType(*E_1*,* E_2*)*
Policy Specifies E_1 **has state** E_2 $\rightarrow$ *hasState(*E_1*,* E_2*)*
Policy Specifies E_1 **role trans** E_2 $\rightarrow$ *roleTrans(*E_1*,* E_2*)*

The predicate of $fact$ in the declarative statement is translated as the predicate of the Datalog literal, and the arguments of $fact$ are translated as arguments of the Datalog literal with translating technique in the proposed language. And as for the declarative statement with the conditional clause as well, we use conditional clause as the body of the Datalog clause after the translation to Datalog literal by the above technique.

B. Verification of Information Flow

In this section, we present the verification of the information flow by the state designation in the proposed language and show the actual description of access control using the result of it.

In the proposed language, we verify the information flow in the future seen from one moment of the process using the reasoning with the embedded predicate **will be authorized**

to. Additionally, we describe an access control rule based on the verification of the information flow which existed in the past, acquiring dynamically from the system, using the the embedded predicate **has state** with user-defined state by which process state designation should be made possible.

We verify the information flow of the future and the past with the following process.

As for the future information flow, DS is verified by evaluating the authorization decision sentence for which the embedded predicate **will be authorized to** was used. The past information flow is deduced by applying the rules (b) $\sim$ (e) in the section III-B recursively to the authorization decision sentence includes a variable clause.

For instance, we assume the DS which keeps the declarative statement of the following 11 rules and show the process which deduces the future information flow from a certain authority set, denoting U for user, R for role, T for type, C for object class and AV for operation.

$$\textbf{Policy Specified } U_A \textbf{ has role } R_A \tag{1}$$
$$\textbf{Policy Specified } U_B \textbf{ has role } R_B \tag{2}$$
$$\textbf{Policy Specified } U_C \textbf{ has role } R_C \tag{3}$$
$$\textbf{Policy Specified } R_B \textbf{ role trans } R_C \tag{4}$$
$$\textbf{Policy Specified } R_A \textbf{ has type } T_A \tag{5}$$
$$\textbf{Policy Specified } R_B \textbf{ has type } T_B \tag{6}$$
$$\textbf{Policy Specified } R_C \textbf{ has type } T_C \tag{7}$$
$$\textbf{Policy Specified } T_A \textbf{ is authorized to } transition\ T_C \textbf{ for } C \textbf{ in } T_B \tag{8}$$
$$\textbf{Policy Specified } T_A \textbf{ is authorized to } allow\ AV_A \textbf{ for } C \textbf{ in } T_B \tag{9}$$
$$\textbf{Policy Specified } T_C \textbf{ is authorized to } allow\ AV_B \textbf{ for } C \textbf{ in } T_D \tag{10}$$
$$\textbf{Policy Specified } T_C \textbf{ is authorized to } allow\ AV_A \textbf{ for } C \textbf{ in } T_B \text{ IF } T_C \textbf{ has state } passed_{td} \tag{11}$$

With this example, when the future information flow of U_A is verified, U_A will be authorized the authority set of AV_A of C in T_B and AV_B of C in T_D by deducing with the rules (1) (5) (8) and (9). And when the future information flow of U_B is verified, U_B will be authorized the authority set of AV_B of C in T_D by deducing with the rules (2) (4) and (10). Similarly in the case of U_C, the authority set of AV_B of C in T_D will be authorized by deducing with the rules (3) (7) and (10). For the verification of the past information flow, we define the state $passed_{td}$ and give the truth value of "whether a certain subject accessed T_D" dynamically from the system. With this example again, fixed authority set for T_B is authorized T_C by evaluating the rules (2) (7) and (11) which designate T_C accessed T_D in the past.

In the proposed language, the information flow can be verified exhaustively by applying deductive rules recursively even when the transition of the roll and the type piles up to many times. Additionally, we can describe various access control rules flexibly in the proposed language using the user-defined state. For example, the access control models of HBAC and execution history based access control can be described using the states of the process and the system by acquiring "process was started.", "process was finished.", and so on dynamically and it can be expected that the Chinese Wall model can be described efficiently using the state of "The resource of a certain domain was read once."

V. Discussion

In this chapter, we present the results of the experiment which evaluates the advantages of the proposed language. First, we show the outline of the experiment system and programs. Then, we compare the response performance to judge authorization with the proposed language and SELinux, examining the results considering the impact of the access vector cache (AVC). And, we confirm the contents of the responses and demonstrate that proposed language can compose the policy of SELinux equivalently when the default policy of actual SELinux is constructed by the proposed language. Finally, we compare the features which can be used for the description about the proposed language and SELinux to evaluate the expressiveness qualitatively.

A. Outline of Experiment System

The experiment system is implemented on the computer of the Pentium 4 (3.0GHz) and the memory 1GB and OS is Debian/GNU Linux added the packages related to SELinux. The versions of main softwares are libselinux1 (2.0.65-5), selinux-basics (0.3.5) and selinux-policy-default (2:0.0.20080702-6).

With the experiment system, we change the source related to SELinux so that we can acquire the address of sidtab and avtab which exist in the kernel space, and get all the contents of the table from the pointer to the acquired hash layout. Then, we change the source to call the function $context\_struct\_compute\_av()$ from the outside of kernel, and measure processing time except for the translation from SID to the security context. Furthermore, we describe all the elements on avtab by the proposed language and compose the policy database using Datalog from the descriptions and prepare the queries by the combination of the source context, the target context and the target class.

We use the RDTSC instruction of x86 for the measurement of the time and devise not to read TSC of another core setting the CPU affinity flag.

B. Evaluations of Response Time and Contents

In this section, we confirm the response performance and the equality of the contents of response describing the default policy of SELinux in the proposed language. In the experiments, we call the function $context\_struct\_compute\_av()$ by using all the elements of sidtab at the round robin to the security class, and measure the time when the acquisition of AV as response. After that, we confirms the equality of the response contents, comparing the responsed AV of

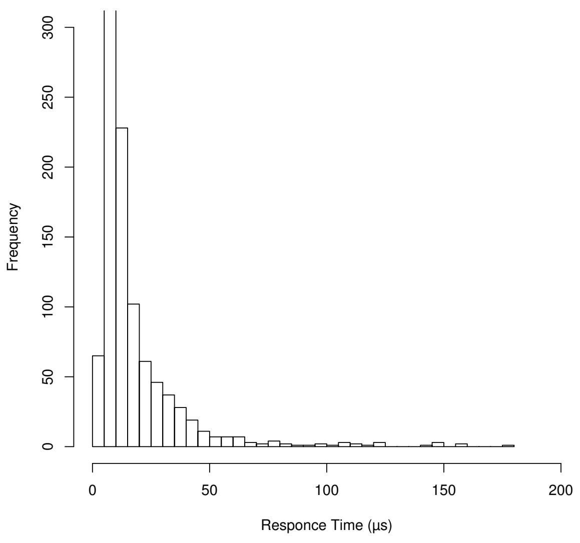

Figure 1. Response Time of SELinux

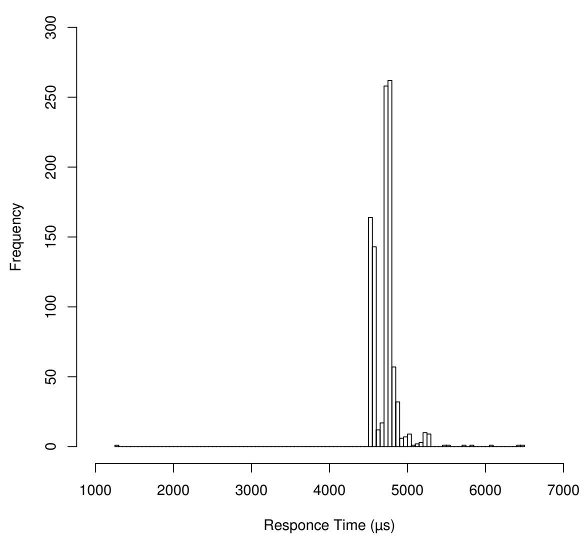

Figure 2. Response Time of Proposed Language with Datalog

$context\_struct\_compute\_av()$ and the result of querying the Datalog with the combination of the same source context, the same target context and the same target class.

We show the result of measurement of the response performance about SELinux and the proposed language in fig.1 and fig.2. Each figure shows the frequency distributions from the result of the experiments for 1,000 elements choosed at random from 13,803,559 elements of all sid which make queries combined at the round robin. The x-axis designates the response time from a demand for a authorization decision to acquisition of AV, and the y-axis designates the frequency of each response time.

As a result, the response times of SELinux were minimum of 2.775μs, maximum of 177.2μs and median of 11.44μs. And the response times of proposed language with Datalog were minimum of 1,275μs, maximum of 6,485μs and median of 4,737μs. Beyond a two-digit number of difference between SELinux and proposed language was confirmed from the experiments. As for the equality of contents of responses, for 13,803,559 queries, same AV of 2,629 acquired and no AV of 13,281,253 acquired which mean the contents of response was the same in SELinux and the proposed language. As a result, about 96% responses were equal in the contents of response. And, the rest of responses were the error by querying to the context which was formed dynamically at the time of the systems operation.

By the experiments, it was demonstrated that all the contents of responses with the same query was corresponded between SELinux and the proposed language with Datalog except for the errors, and confirmed that the policy of SELinux could be expressed equivalently in the proposed language. On the other hand, the response time of the proposed language was inferior greatly for the query of the same contents. We guess the inferior is caused by the difference of the implementing form and the search algorithm of the policy database. About the implementing form, the policy database and the evaluation functions are implemented inside the kernel space for SELinux, while the proposed language with Datalog implements them in the user space. Hence, for improvement, the processing system of Datalog need to be set inside the kernel and the execution steps concerned with the context switch can be reduced. Additionally, about search algorithm, we can guess that this difference is reduced because almost all queries are processing by the cache mechanism called AVC in actual operations, and it is said generally the AVC hit rate of SELinux is over 99%.

C. Evaluation of Expressiveness

In this section, we compare the features of the proposed language and SELinux, evaluating the expressiveness qualitatively. Three functions used for policy descriptions are evaluated. Table I is the results.

The first function is to make the rule enable/disable based on some conditions. This means the result of not only searching the existence of the queried statement in the policy but also evaluating the various conditions given to each rule decide the authorization decision. The expression of flexible authorization decisions using the second function and the third function is achieved by using the first function. For example, by the former technique, rules become the forms

Language / Features	SELinux Policy	Proposed Language
1. Enabling/Disabling the rules based on the condition	△	○
2. Treating the system or process state as the condition	X	○
3. Treating the future/past information flow as the condition	X	○

Table I
COMPARISON THE FEATURES OF DESCRIPTION

of "allow the process A to read the resource B." On the other hand, by the latter technique, rules become the forms of "If the condition C is true, allow the process A to read the resource B.", introducing condition evaluation for the rule, therefore, the rules are enable/disable depending on the results of the evaluation of the condition.

The second function is to describe systems and processes states as a condition and the state transition system can be expressed efficiently by this function. Furthermore, we can describe the authorization rules considering the transaction processing states, the damage states, and the time states, and so on by this function. For example, the rules like followings can be expressed; "If the process A and B is in state 'finished', allow the process C to read the resource D." for the transaction processing state, "If the process A is in state 'polluted', deny the process B to read the resource C-F." for the damage states, "If the system is in state 'AfterDate20090815', allow the process A to read the resource B." for the time states.

The third function is to describe the result of information flow verification as a condition, and the rules based on the temporal logic can be expressed by this function. For example, the rules like followings can be expressed; "If it would be authorized to write the resource A in the future, reading of the resource B can not be authorized.", ""If it has writing authority to the resource A from now on, reading to the resource B will be recognized.", "If the resource A had been read in the past, writing to the resource B would not be authorized."

VI. CONCLUSION

In this paper, we proposed a policy description language which can describe dynamic access control models in addition to the existent models like RBAC and DTE to prepare for the application range expansion of Multi-Layered Defense. We showed the description technique and some deductive rules which make each access control rule effective evaluating the dynamic information. And we implemented the proposed language using Datalog, and explain the concrete way of describing the access control rule by the proposed technique which is based on the future/past information flow and state designation of the system and the process. Finally, we confirmed the advantages of the proposed language by evaluating response times, response contents and expressions by the experiments. The improvement of the performance and the formal proof of the proposed language are the future works.

ACKNOWLEDGMENT

The authors are grateful to the anonymous referee for her/his valuable comments.

REFERENCES

[1] "Trusted computer systems evaluation criteria," DoD Computer Security Center, Fort Meade, MD, Tech. Rep. CSC-STD-001-83, August 1983.

[2] S. Ceri, G. Gottlob, and L. Tanca, "What you always wanted to know about datalog (and never dared to ask)," *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, no. 1, pp. 146–166, Mar 1989.

[3] D. H. D. Warren, L. M. Pereira, and F. Pereira, "Prolog - the language and its implementation compared with lisp," in *Proceedings of the 1977 symposium on Artificial intelligence and programming languages*. New York, NY, USA: ACM, 1977, pp. 109–115.

[4] R. Spencer, S. C. Corporation, S. Smalley, P. Loscocco, N. S. Agency, and M. H. D. Andersen, "The flask security architecture: System support for diverse security policies," in *in Proceedings of The Eighth USENIX Security Symposium*, 1999, pp. 123–139.

[5] M. Abadi, "On sdsi's linked local name spaces," *J. Comput. Secur.*, vol. 6, no. 1-2, pp. 3–21, 1998.

[6] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen, *SPKI Certificate Theory*. United States: RFC Editor, 1999, ch. 2693.

[7] M. Blaze, J. Feigenbaum, and M. Strauss, "Compliance checking in the policymaker trust management system," in *FC '98: Proceedings of the Second International Conference on Financial Cryptography*. London, UK: Springer-Verlag, 1998, pp. 254–274.

[8] M. Blaze, J. Feigenbaum, and A. D. Keromytis, "Keynote: Trust management for public-key infrastructures (position paper)," in *Proceedings of the 6th International Workshop on Security Protocols*. London, UK: Springer-Verlag, 1999, pp. 59–63.

[9] M. Becker, C. Fournet, and A. Gordon, "Design and semantics of a decentralized authorization language," in *CSF '07: Proceedings of the 20th IEEE Computer Security Foundations Symposium*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 3–15.