Computing and Informatics, Vol. 25, 2006, 353–368

FORMAL VERIFICATION OF SECURITY MODEL USING SPR TOOL

Il-Gon Kim, Miyoung Kang, Jin-Young Choi*

Korea University
Seoul, Korea
e-mail: igkim@formal.korea.ac.kr

Peter D. Zegzhda, Maxim O. Kalinin, Dmitry P. Zegzhda

Saint-Petersburg Polytechnical University
Saint-Petersburg, Russia

Inhye Kang

University of Seoul
Seoul, Korea

Manuscript received 3 June 2004; revised 12 June 2006
Communicated by Igor Walukiewicz

Abstract. In this paper, formal verification methodologies and the SPR (Safety Problem Resolver) model checking tool are used for verifying a security model's safety. The SPR tool makes it possible to analyze security issues on security systems based on the access control model. To illustrate this approach, a case study of the Simple Access Control Model (SACM) is used and specific safety problems of the security model are analyzed using the SPR tool.

Keywords: SPR (Safety Problem Resolver), SEW (Security Evaluation Workshop), SPSL (Safety Problem Specification Language)

* Corresponding author

1 INTRODUCTION

The greater the assurance, the greater the confidence that a security system will protect against threats, with an acceptable level of risk. A contrast is revealed in supposedly secure commercial operating systems, applications, and network components, particularly with respect to security. Commercial offerings have serious security vulnerabilities. Most existing systems lack adequate ability to securely interconnect and interoperate. Each vendor has taken a different approach, demonstrating relative independence from their competitors. The revealing of all vulnerabilities comprises the goal of the security evaluation process. There is little understanding as to the level of security that can be attained by integrating a collection of components, and even less understanding as to what assurance security may provide.

In the last decade, a number of individual countries developed specific security evaluation standards [16]. This initiative opened the path to worldwide mutual recognition of security evaluation results. Following this initiative, a new Common Criteria (CC) [3] were developed. For example, CC defines seven levels of assurance for security systems, rising from EAL1 to EAL7. In order to obtain a higher assurance than EAL5, developers require specification of a security model for security systems and verify security using formal methods approach.

Vendors are discouraged from offering secure systems because significant resources are required to develop a system capable of meeting the evaluation criteria and to marshal it through the evaluation process. Moreover, because of evaluation delays, an evaluated product is typically no longer the current version of the system, which necessitates repeated reevaluation. For high assurance systems, the difficulty of using formal methods adds further complexity to both development and evaluation. However, given the lack of suitable mature, industrial-strength tools, and the cost of formal verification, an informal approach generally presents a suitable compromise.

This paper proposes a technique for analysis of security policy enforcement, a framework for formal analysis of security systems, and Safety Problem Resolver (SPR) [10]. These allow specification of system security-related elements and verification of system safety.

This paper is structured as follows. Section 2 addresses the classification of security models and illustrates the background of issues. Section 3 introduces the overall structure of the Safety Evaluation Workshop (SEW), in analyzing the safety of a security model. Section 4 presents core specification and verification, illustrates SPSL, and presents methods for resolving safety problems using the SPR tool. In Section 5, an example of formal specification and verification for the Simple Access Control Model (SACM) is presented using the SPR tool. Finally, Section 6 presents the conclusion and discusses future direction.

2 SAFETY PROBLEM

In general, *security* [14] represents the combination of confidentiality, integrity and availability. The *security model* term [13] could be interpreted as the formal representation of a security system's confidentiality, integrity and availability requirements. The more general usage of the term specifies a particular mechanism for enforcing confidentiality and named access control, which has been adopted for computer security from the world of documents and safes.

Security concerns arise in many different contexts; thus, many security models have been developed. The access control model can be grouped into three main classes according to security policies [6]: Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role Based Access Control (RBAC).

Obtaining assurance that system behavior will not result in unauthorized access is called a **safety problem** [13]. MAC governs access on the basis of classification of subjects and objects in the system. MAC is safe by definition, and no safety problem resolving is required for the MAC-based system, because safety for MAC is proven theoretically and presented in a general case [12]. The verification of the safety of any kind of MAC policy is obvious; however, unfortunately, the policies for MAC are not particularly well suited to industry organizations that process sensitive information.

DAC has the drawback that it does not provide real assurance on the flow of information in a system. It is easy to bypass stated access restrictions, through another user's authorization process. For example, a user permitted to read data can pass this data to other unauthorized users without the cognizance of usage of information by a user once the user receives the information.

Since Harrison, Ruzzo, and Ullman demonstrated that the safety problem was undecidable, in the common case [7], research has focused on determining whether safety could be decided for access control models with limited, but practical, expressive power. First, the take-grant model has a linear time safety algorithm; however, there is still a significant difference in expressive power between take-grant and HRU [1]. This difference is considerably smaller for the models presented by Sandhu et al.: SPM, TAM, ESPM, and non-monotonic ESPM [2]. Sandhu et al. demonstrated that an access control model could be designed for which safety is efficiently decidable (i.e., in polynomial time), given some restrictions. Ultimately, despite proven expressive power and safety determination, these access control models have not been adopted in practice. In this paper, we give two reasons for this lack of acceptance:

1. subtlety of the restrictions and complex relationship and
2. difficulty in defining safety requirements and writing practical algorithms to enforce these requirements.

With RBAC [5], access decisions are based on roles that individual users have as part of an organization. Users take on assigned roles. In spite of the mentioned

lacks, great majority of systems (e.g. operating systems, DBMS, Firewalls) use DAC-based security models as the basis for access control mechanism. Thus, in a general situation, safety cannot be verified for the arbitrary DAC access control system. In security evaluation, safety verification is the central problem, particularly for distributed computer systems.

In this paper, this safety problem is solved, proposing a universal specification and model checking tool. This tool permits the analyzer to describe system security elements and calculate security estimation for any security computer system.

3 SAFETY EVALUATION WORKSHOP

According to principles of computer system modeling, the term *security model* is used as the combination of system security states, transitions through access control rules, and constraints such as the state security criteria.

Figure 1 demonstrates three components of the formal *security model*. These mainly present access control models [7, 12, 6]. The access control model was first formulated by Lampson [11]. The structure of the model is that of a state machine where each state is (S, O, M), where S is a set of subjects, O is a set of objects, and M is an access matrix which has one row for each subject, one column for each object, and is such that cell $M[s, o]$ contains the access rights.

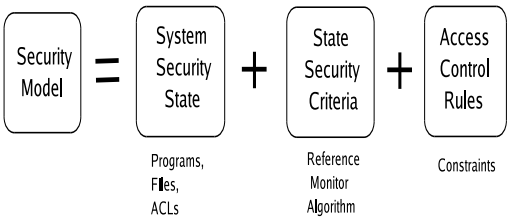

Fig. 1. Security model structure

The type of security system mentioned in this paper, refers to operating systems, IDSs, Firewalls, and so on. For safety evaluation of security systems, a Safety Evaluation Workshop (SEW) structure is proposed, consisting of seven components. Figure 2 presents the evaluation framework in SEW.

The functions of seven components in SEW are as follows:

1. System State Analyzer:

   Investigates the system being evaluated and builds the security model state automatically according to the access control model.

2. Security Criteria Manager:

   Evaluator inputs the state security criteria into the Security Criteria Manager.

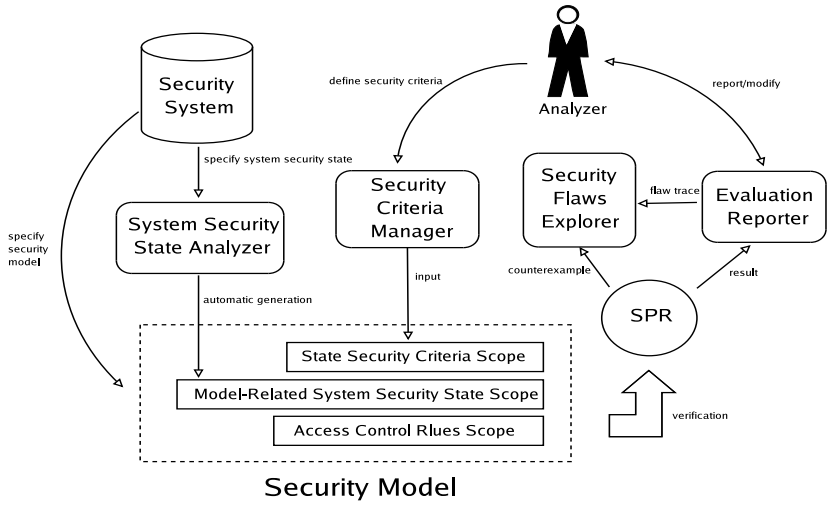

Fig. 2. Safety Evaluation Workshop structure

3. Scopes:
   (a) Model-related System Security (M3S) – scope:
       Specifies the system security state and behavior in SPSL.
   (b) Access Control Rules (ACR) – scope:
       Describes access control rules in SPSL.
   (c) State Security Criteria (SSC) – scope:
       Expresses state security criteria in SPSL.
4. SPR:

   Formal verification tool implemented using C and SWI-Prolog [17].
5. Safety Problem Specification Language (SPSL):

   Specification language for security model with 3 scopes, based on Prolog syntax.
6. Security Flow Explorer:

   Demonstrates the sequence of the events, leading to the security fault.
7. Evaluation Reporter:

   Produces a final report containing the access control model, system, initial state, access control rules, security criteria, evaluation result, and security flaw trace.

4 SAFETY PROBLEM RESOLVING AND SPSL

4.1 General System

A general system Σ could be represented as a state machine: $\Sigma = \{S^\Sigma, T, s^\Sigma_{init}, Q\}$, where S^Σ is a set of system state; Q is the set of the queries executed by the system, such as create file, read file or execute file; T is the state transition function, $T : Q \times S^\Sigma \longrightarrow S^\Sigma$ moves the system from one state to another: a request q is issued in the state s^Σ_i and moves the system to the next state $s^\Sigma_{i+1} = T(q, s^\Sigma)$; s^Σ_{init} is the initial system state.

4.2 General Security Model

A general security model M presents an ensemble of sets, $M = \{S, R, C\}$, where S is the set of system security states defined by the model; R is the set of the access control rules; C is the set of the state security criteria in the form of logical predicates $c(s)$ defined on S and checking security of the state s. A state $s \in S$ is secure if and only if every security criterion c from C the predicate $c(S)$ is true.

4.3 Safety Property

The term of *safety* means that "something bad never happens". If a security system has safety problem in security, it represents that sensitive information is leaked by unauthorized access. The assurance that system behavior will not result in unauthorized access is fundamental to ensuring that the enforcing of the access control model will guarantee system security.

The system safety property can be formalized as $\Lambda = \{M, \Sigma, D\}$, where M is the access control model, $M = \{S, R, C\}$; Σ is the system, $\Sigma = \{S^\Sigma, T, s^\Sigma_{init}, Q\}$; D is the mapping function, $D : S^\Sigma \longrightarrow S$, which is the relationship between the system states and system security states.

4.4 SPSL

In order to specify the system security-related elements, in this paper the Safety Problem Specification Language (SPSL) is presented. The majority of other work relating to security specification is not intuitive and does not easily map onto tasks.

The ASL [9] represents an example technique based on the formal logic language for security specification. Although support is provided for role-based access control, the language does not scale well in real systems because there is no method of modeling real access control rules. There is no specification of delegation and no method of modeling rules for groups of different identities. LaSCO [8] is a graphical approach for evaluating security constraints on objects, in which the policy consists of two components: domain (the system abstraction) and requirements (authorization rules). The scope of this approach is excellent for plain demonstration, but too

complex for actual implementation. The Ponder [4] language is an object-oriented specification language. This language is most effective for safety specification and evaluation purposes. This language is developed using JAVA implementation, but is not well prepared for automated evaluation of real-world systems. The Ponder-based system does not support system security state modifications and state transitions.

Therefore, this general evaluation problem for security policy enforcement, including system weakness detection, has never been addressed by any security specification tools. SPSL represents a logical specification language to express a Model-related System Security Scope (security states), Access Control Rules Scope (access control rules), and Security Criteria Scope (security criteria) based on the Prolog-style syntax (SWI-Prolog usage). SPSL is designed to specify the protection mechanisms of common-used operating systems such as Windows 2000 and Linux. For example, consider the Windows 2000 operating system. In this system, system security is configured by subject (e.g., users, user groups), objects (e.g., files, registry keys), and access control rights. The access control rights are composed in a list, called the Access Control List (ACL). There is a map between subjects, objects, and access modes in the ACL.

4.5 Safety Problem Resolving in SPR

In this paper, the Safety Problem Resolver (SPR) tool [10] is proposed, which helps analyze and process SPSL-based specifications regarding the system security, verifying whether a system security policy has a safety problem. The SPR verifies an initial system security state using a given security criteria; first all reachable states are generated, then they are verified using the criteria. The SPR tool is a logical machine based on the Prolog kernel, supplied with an API implemented in the C++ programming language. The SPR can be applied to safety problem resolving as described below.

For analyzing safety problems using SPR, it is necessary to execute three stages.

1. Evaluate whether the given initial system $s^{\Sigma}_{init} \in s^{\Sigma}$ state conforms to the security criteria C:
$$\forall c \in C : c(D(s^{\Sigma}_{init})) = \text{true}.$$
2. Analyze the system access control mechanism that realizes the access control rules R:
$$\forall s^{\Sigma}_{i}, s^{\Sigma}_{i+1} \in S^{\Sigma} : s^{\Sigma}_{i+1} = T(q, s^{\Sigma}_{i})$$
$$\exists s_i = D(s^{\Sigma}_{i}), s^{\Sigma}_{i+1} = D(s^{\Sigma}_{i+1}) \text{ and } \forall r \in R : r(s_i, s_{i+1}) = \text{true}.$$
3. Generate the states $s^{\Sigma}_{i} \in s^{\Sigma}$ reachable from the given initial state $s^{\Sigma}_{init} \in s^{\Sigma}$ and analyze their safety by the security criteria C:
$$\forall s^{\Sigma}_{i} \in S^{\Sigma} : s^{\Sigma}_{i} \text{ is reachable from } s^{\Sigma}_{init}, s^{\Sigma}_{i} = D(s^{\Sigma}_{i}) : \forall c \in C : c(s_i) = \text{true}.$$

If the above conditions are met, the system in the given system state is safe according to the access control model. The process of producing sets of reachable states and evaluating the criteria is called *safety problem resolving*.

The SPR tool searches all reachable states in systems and verifies whether they represent an unsafe security state. A more detailed description of the SPR is presented in [15].

5 SECURITY EVALUATION EXAMPLE

For easy understanding of security specification and SPR's functionality, a very Simple Access Control Model (SACM) example is presented in Figure 3.

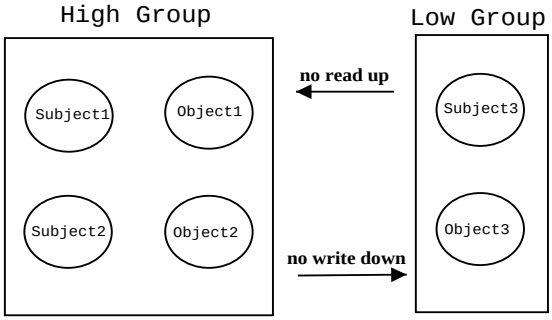

Fig. 3. Simple Access Control Model example

The access control model presented in Figure 3 has High and Low groups. Subject1, Subject2, Object1 and Object2 are in the High group. Subject3 and Object3 are involved in the Low group. In this example, a Subject refers to a user, and an Object points to a file. This example access control model should satisfy the following safety properties.

1. No Read Up: A subject is allowed read access to an object only if the access class of the object dominates the access class of the subject.
2. No Write Down: A subject is allowed write access to an object only if the access class of the subject is dominated by the access class of the object.

The satisfaction of these principles prevents information in high level objects to flow to objects at lower levels.

5.1 Security States

The *security states* are a collection of all entities of the system (subjects, objects) and their security attributes (access rights, ACLs, and etc.), similar to well-known operating system. In this example model, it is assumed that a subject has each

Subjects \ Objects	Object1	Object2	Object3
High Group	read,write	read,write	read,write
Low Group			read,write
Subject1	read,write	read,write	read,write
Subject2	read,write	read,write	read
Subject3			read,write

Table 1. Simple Access Control Model's ACLs

object in its directory. The SACM includes user groups and files. User groups are divided into High and Low ones and are noted as security subjects. Subject1, Subject2 and Subject3 represent three users. File system elements(files and directories) are regarded as security objects. Object1, Object2 and Object3 are noted as 3-scope files in each user's directory. The object access by subject depends on the ACLs shown in Table 1. The system security states may be presented in terms of Model-related System Security Scope (M3S)-scope; Subjects + Objects + Security Attributes.

Example 1. M3S-scope of SACM

```
subjectAttr(subjectGroups).
subject(s1,[subjectGroups(high)]).
subject(s2,[subjectGroups(high)]).
subject(s3,[subjectGroups(low)]).
objectAttr(objectType).
objectAttr(high).
objectAttr(low).
objectAttr(s1).
objectAttr(s2).
objectAttr(s3).
object(o1,[objectType(file), objectOwner(s1), high(rd,rp,wd,wp),
       low, s1(rd,rp,wd,wp), s2(rp,rd,wd,wp), s3]).
```

Example 1 presents some components of M3S-scope in SPSL. According to the form of M3S-scope mentioned above, SACM's initial security state using SPSL is specified based on Figure 3 and Table 1. This consists of three elements;

- Subjects and their membership in the group (e.g., *subjectGroups*).
- Objects and their hierarchy (e.g., *objectType* and *objectOwner*).
- Initial values of the subjects' and objects' security attributes (e.g., *subjectAttr*, *subject*, *objectAttr* and *object*).

Predicate *subjectAttr* represents a subject's security attribute. Attribute *subjectGroups* depicts user membership in the groups. Thus, the statement 'subjectAttr(subjectGroups)' in the first line means to create a new type of *subjectGroups*.

With a predicate *subject* declared as an initial set of the subjects in the SACM. The predicate *subject* represents two parameters, the name of the subject and its attribute values. For example, the statement 'subject(s1,[subjectGroups(high)])' in the second line means that "*s1* is a subject in a high group". With predicate *objectAttr*, a set of the objects security attributes is declared. The *objectType* attribute in the Prolog list of predicates depicts the type of objects and is generally used to distinguish whether the object is a file or directory. Attribute *objectOwner* describes the owner of object.

Then, there are five attributes with names of users and groups. In a predicate *object*, an initial set of objects are declared in SACM. The predicate *object* has two parameters, name of the object and its attribute values. Therefore, the 12^{th} line in Example 1 can be interpreted as follows: "The object of *o1* is a file. The owner of object *o1* is *s1*. The high group has an access permission right of *rp*, *rd*, *wd*, *wp* to *o1*. The low group possesses no access rights to *o1*. The subject of *s1* possesses access rights of *rd*, *rp*, *wd*, *wp* to *o1*. The subject of *s2* possesses access rights of *rd*, *rp*, *wd*, *wp* to *o1*. The subject of *s3* has no access rights to *o1*." The privilege *rd* allows reading directory entries such as listing files and subdirectories. The function of *rd*, *rp*, *wd*, and *wp* privileges is presented in Section 5.2. The object states of *o2* and *o3* are not described because it is trivial to obtain the associated SPSL code from Table 1, Figure 3, and the example of the *o1* object.

Using a similar structure, any real system securable object can be specified. In order to automate this approach a special tool has been developed, e.g., State Analyzer for Windows, that investigates the system and creates the system security state.

5.2 Access Control Rules

Access control rules express the restrictions on system behavior. System state transformation is possible after access is authorized by the system reference monitor (access control mechanism). Authorization is checked against the security policy requirements represented using access control rules. In the SACM example, a subject can have actions *rd*, *rp*, *wd* or *wp*. The *rd* command allows reading directory entries, i.e., listing of files and reading data stored in file. The *rp* command allows reading a files privileges. The *wd* command allows file creation in a directory. The *wp* command allows modification of a files privileges of the file. Such specification can be called Access Control Rules (ACR)-scope. Example 2 shows some components ACR-scope of SACM. Due to space and coverage limitations in this paper, full ACR related information is not described in SPSL. In fact, it is necessary to investigate a number of manuals, specifications, APIs, and vulnerability exploits to obtain ACR-scope for analyzing real operating systems such as Windows 2000.

Example 2. ACR-scope of SACM

```
validPermission(R):-
  (R=rd;R=wd;R=rp;R=wp).

isGroupMember(S,G):-
   validSubject(S),
   validGroupName(G),
   checkAttr(S,subjectGroups,G).

isFile(O):-
   validObject(O),
   checkAttr(O,objectType,file).

canReadFile(S,O):-
   validSubject(S),
   isFile(O),
   pReadData(S,O),
   canReadPermissions(S,O).

canWriteFile(S,O):-
   validSubject(S),
   isFile(O),
   canReadFile(S,O),
   pWriteData(S,O).

.........[abbreviation].........

testState1(S,O):-
    validSubject(S),
    isFile(O),
    canReadFile(S,O),
    not(isGroupMember(S,high)),
    O=o1,
    O=o2.

testState2(S,O):-
    validSubject(S),
    isFile(O),
    canWriteFile(S,o3),
    not(isGroupMember(S,low)).
```

The SPSL source code shown above can be divided into two categories: The first is for security policy requirements, and the second defines predicate prototypes for checking security criteria. With the predicate *validPermission*, access permission R defined in ACLs can be expressed. Predicate *isGroupMember* which verifies S (subject) is a member of G (group). The *isFile* predicate tests whether O (object) is

a file. The predicate *canReadFile* verifies that S (subject) can read from file O. Similarly, *canWriteFile* verifies that S (subject) can write to file O. Predicates *TestState1* and *TestState2* predicates represent security criteria clauses for security policy requirement. *TestState1* depicts "*No Read Up*" policy and *TestState2* denotes "*No Write Down*" policy.

5.3 Security Criteria

The *security criteria* allows a customer or evaluator to verify the secure and insecure states in a security model. The security criteria have a form of constraints which state the necessary conditions of the secure system. The system is safe if $\bigcap_{i \in N} \overline{cr}_i = true$, where cr_i is an undesirable state criterion. The notation of $\overline{cr}_i$ is the negation of cr_i. In other words, $\overline{cr}_i$ which means an unsafe state should not be found in a security model. The SPR tool's security criteria can be represented as State Security Criteria (SSC)-scope. Example 3 demonstrates SSC-scope of the SACM written in SPSL.

Example 3. SSC-scope of SACM

```
testState1(_,_). testState2(_,_).
```

Two predicates, *testState1(_,_)* and *testState2(_,_)* call the criterion predicate shown in Example 2. In Prolog language, '_' means the sign of an anonymous variable.

5.4 Safety Evaluation Results Processing

SPR input with triple scopes(M3S-scope, ACR-scope, and SSC-scope) are written in SPSL. The executable program for SACM is then executed. During execution, two files are created: the security logical deduction trace (SRP.TRC) and the evaluation report (SPR.REP). Example 4 presents the SPR.TRC file for SACM. With regard to space considerations, only the shorter traces are shown. This raw format content represents Prolog backtracking traces.

Example 4. SPR.TRC of SACM

```
3 call system:not(testState1(_G7, _G8))
4 call testState1(_G7,_G8)
5 call isFile(_G8) 8 call system:
  (objectType(file)= ..[objectType|_G136])

..............[abbreviation]..............

1 fail spr
2 call system:retractall(spr)
2 exit system:retractall(spr)(objectType(file)
  = ..[objectType|_G136])
```

Example 5. SPR.REP of SACM

```
/*
* SPR report file
* File contains its results about safety
*/
    testState1(_,_)         succeeded
    testState2(_,_)         failed
```

Example 5 presents the output of file SPR.REP. The result for checking the *testState1* criteria is "succeeded". This means there is no subject in the low group, which can read objects in the high group. The evaluation verdict for *testState2* is "failed", i.e. the SACM initial state is unsafe. After analyzing this unsafe state, *s1* in High group can be found to have an access right permission of *wd, wp* to *o1* in a low group. The incorrect setting of ACLs for subjects and objects permits the SACM to violate associated security requirements. For the safety of the SACM, Subject1 is set to only have read permission for Object3. The modified predicate can be written as follows:

```
object(o3,[objectType(file),objectOwner(s3),high(rd,rp),
low(rd,rp), s1(rd,rp),s2(rd,rp), s3(rd,rp,wd,wp)]).
```

6 CONCLUSION

In this paper, we addressed formal specification and verification approach for a security model. The SPR model checking tool and its associated method for safety problem reasoning have been presented. The tool permits verification of safety properties in terms of the security model, based on the security scope. After conducting the case study of analyzing safety problems in the SACM, the SPR tool is confirmed to be useful for security system customers and evaluators, because these kinds of tools are rare in the security field. The targets of this application include computer systems based on state machine presentation: operating systems, DBMSs and Firewalls. The current versions of these tools are aimed at well-known systems such as Microsoft Windows NT series and UNIX based systems. For future work, the SEW components including System State Analyzer, Security Criteria Manager, and Security State Explorer will be developed to support straightforward modeling and analysis for system security and safety problems.

REFERENCES

[1] Bishop, M.—Snyder, L.: The Transfer of Information and Authority in a Protection System. In Proceedings of the 7th ACM Symposium on Operating Systems Principles, 1979, pp. 45–54.

[2] Castano, S.—Fugini, M. G.—Martella, G.— Samarati, P.: Database Security. Addison-Wesley, 1995.

[3] National Institute of Standards and Technology, Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and General Model, Version 2.1. CCIMB-99, August 1999.

[4] Damianou, N.—Dulay, N.—Lupu, E.—Sloman, M.: The Ponder Policy Specification Language. Proc. Policy 2001: Workshop on Policies for Distributed Systems and Networks, Bristol, UK, 2001.

[5] Ferraiolo, D.—Kuhn, R.: Role-Based Access Controls. In Proc. of the 15th NIST-NCSC National Computer Security Conference, Baltimore, 1992, pp. 554–563.

[6] Goguen, J.—Meseguer, J.: Security Policies and Security Models. In Proceedings of the 1982 IEEE Symposium on Research in Security and Privacy, 1982.

[7] Harrison, M. H.—Ruzzo, W. L.—Ullman, J. D.: Protection in Operating Systems. Communications of the ACM, 1976, pp. 461–471.

[8] Hoagland, J. A.—Panday, R.—Levitt, K. N.: Security Policy Specification Using a Graphical Approach. Tech. report CSE-98-3, UC Davis Computer Science Dept., 1998.

[9] Jajodia, S.—Samarati, P.—Subrahmanian, V. S.: A Logical Language for Expressing Authorizations. Proc. of the IEEE Symposium on Security and Privacy, Oakland, CA, 1997, pp. 94–107.

[10] Il-Gon, K.—Jin-Young, C.—Zegzhda, P. D.—Kalinin, M. O.—Zegzhda, D. P.—Inhye, K.—Pi-Yong, K.—Wan, S. Yi.: Analyzing the Safety Problem in Security Systems Using SPR Tool. PARA '04 Workshop, June 2004.

[11] Lampson, B. W.: In 5th Princeton Symposium on Information Science and Systems. pp. 437–443, Reprinted in ACM Operating Systems Review, 1974, pp. 18–24.

[12] LaPadula, L. J.—Bell, D. E.: Secure Computer Systems: A Mathematical Model. Technical Report ESD-TR-278. Vol. 2, The Mitre Corp., Bedford, MA, 1973.

[13] McLean, J.: Security Model. In Encyclopedia of Software Engineering, Wiley Press, 1994.

[14] Shirey, R.: RFC 2828. Internet Security Glossary, Networking Working Group, 2000.

[15] White Paper,
Available on: `http://www.ssl.stu.neva.ru/spr/eng/whitepaper.htm`.

[16] Department of Defense, Trusted Computer System Evaluation Criteria, DOD 5200.28-STD, Dec 1985.

[17] Wielemaker, J.: SWI-Prolog 5.2 Reference Manual.
Available on: `http://swi-prolog.org`, July 2003.

Il-Gon Kim is a postdoc researcher in IRISA/INRIA. He received M. Sc. and Ph. D. degrees in the Department of Computer Science & Engineering from Korea University. His research interests are formal methods, process algebras, security systems, and web services.

Miyoung Kang is a Ph. D. student in the Department of Computer Science and Engineering in Korea University. She received M. Sc. degree in computer science and engineering in Dongguk University. Her research interests are formal methods, static analysis, security and software engineering.

Jin-Young Choi is a professor in the Department of Computer Science and Engineering in Korea University. He received B. Sc. degree in computer engineering, Seoul National University in 1982. He received M. Sc. degree in computer science, Drexel University in 1986 and acquired Ph. D. degree in computer science from the University of Pennsylvania in 1993. His research interests are real-time computing, formal methods (formal specification, formal verification, model checking), process algebras, security and software engineering.

Peter D. Zegzhda is a professor and a chief of the Chair of Information Security in Computer Systems in Saint-Petersburg Polytechnical University, Russia. He was graduated as the D. Sc. at the Saint-Petersburg Polytechnical University in 1996. He is a Director of the Software Security Laboratory in Saint-Petersburg since 1992. His research fields are security modeling, secure operating systems, security standards, code analysis, security architectures, network security.

Dmitry P. Zegzhda is a professor at the Chair of Information Security in Computer Systems, Saint-Petersburg Polytechnical University, Russia. He received the DSc degree in Saint-Petersburg Polytechnical University in 2003. Since 2004, he is the director of LG-Polytechnic Research and Educational Centre, a joint Korean-Russian company. He is a direction manager in the Security Laboratory in Saint-Petersburg. His research interests are secure operating systems, computer viruses, vulnerabilities analysis, network security, intrusion detection systems, code analysis, databases security.

Maxim O. Kalinin is an associate professor at the Chair of Information Security in Computer Systems, Saint-Petersburg Polytechnical University, Russia. He was graduated as Ph. D. at the Saint-Petersburg Polytechnical University in 2003. He started as a worker from the R & D Engineer to the Project Manager in the Software Security Laboratory, Saint-Petersburg. His research interests are formal methods, AI, security modeling, safety evaluation, secure operating systems, vulnerabilities detection, code analysis, mobile security, and multimedia security.

Inhye Kang is an assistant professor at the University of Seoul. She received the Ph. D. from the University of Pennsylvania. Before joining the University of Seoul, she was a visiting professor at Soongsil University from 1998 to 2000, and a chief engineer at Samsung SECUi.com from 2000 to 2002. Her research interests include software engineering, formal methods, and internet security.