An Extension of Role Based Access Control for Trusted Operating Systems and Its Coloured Petri Net Model

Wook Shin

Department of Information and Communications

Gwangju Institute of Science and Technology

2005
안전한 운영체제를 위한 역할기반 접근통제의 확장 및 컬러드 페트리넷 모델에 관한 연구

신옥

2005
An Extension of Role Based Access Control for Trusted Operating Systems and Its Coloured Petri Net Model

Advisor: Hong Kook Kim

by

Wook Shin

Department of Information and Communications

Gwangju Institute of Science and Technology

A thesis submitted to the faculty of the Gwangju Institute of Science and Technology in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Information and Communications

Gwangju, Korea

December 6, 2004

Approved by

Professor Hong Kook Kim

Thesis Advisor
An Extension of Role Based Access Control for Trusted Operating Systems and Its Coloured Petri Net Model

Wook Shin

Accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

December 6, 2004

Thesis Advisor
Prof. Hong Kook Kim

Committee Member
Prof. R.S. Ramakrishna

Committee Member
Prof. JongWon Kim

Committee Member
Prof. Igor Lemberski

Committee Member
Prof. Kouichi Sakurai
To my parents.
Abstract

The notion of trusted operating system was proposed to solve the security problems of current security solutions. Such security solutions as intrusion detection systems, firewalls have been used widely. However, they are run in the application-level of computer systems. Therefore many application-level processes including hacking programs can easily bypass them. Numerous bypassing techniques already have been released, and they reveal the weakness of those solutions.

On the contrary, a trusted operating system provides security mechanisms at its kernel that is the lower level than the application-level. The application-level processes cannot bypass the kernel-level security mechanisms. In addition, the lower-level security mechanisms are more efficient and easier for analysis. Consequently, the trusted operating systems provide solider and more fundamental security mechanisms than application-level solutions.

The security services of trusted operating systems are provided with access controls as the core function. However, the core security mechanism of trusted operating systems is not still sufficient to protect the systems from various attack trials. Some kinds of attacks are successfully executed on the security kernel of trusted operating systems. The insufficiency is originated from the limited functionality of access controls. Current access control schemes have a common limitation in their functions. They cannot effectively protect a system from the attacks that consist of ordinary operations. The access control process gathers access control information at the very time of the access occurring, uses the information to decide legality of the access, and finally discards the information. The access control information is extracted and used instantly, and any
associated information in hidden information between accesses is not considered. This
processing pattern brings functional limitation so that current access control cannot
detect and deny a set of operation set.

In this dissertation, we discuss such problems of access control and introduce an ex-
tension approach to the access control to solve the problem. The research achievements
are presented as follows:

First, we extend Role Based Access Control and compose an advanced access control
scheme for the trusted operating systems. We introduce two properties into the access
control entities: order and negative permission. The order is given to the set of access
objects, and then we can regard the set of operations as a form of procedure. The
negative permission is added to a procedure to express a harmful set of operations.
With the extension, we express the execution sequences of system operations as the
ordered sets, and then control accesses based on not only the traditional access matrix
information, but also the sequence information. As a result, it is possible to detect and
deny the attacks that consist of ordinary operations.

Second, we propose a formal model for the extended access control. The formal
model is intermediary for the correct implementation of the access control policy. More-
over, the formal model is helpful to test correctness of security configuration of a system.
In general, it is hard to find errors in the configuration by manual approach, but formal
methods provide computerized simulation, modeling, and verification tools. Though
there have been formal specification methods for previous Role Based Access Control,
they are not appropriate to describe the execution sequences of extended access con-
trol. Therefore, we suggested formal model based on Coloured Petri Net. With the
Coloured Petri Net formalism, we can express execution sequences as a form of subnet,
and also express access matrix information as the other Coloured Petri Net components.
We can specify the extended access control system with the proposed Coloured Petri
Net based specification model. We can simulate and verify security-related properties
by automated tools. It helps security administration of the extended access control
system.

Third, a simple implementation result of the proposed access control scheme is pre-
sented in this dissertation. The proposed scheme was implemented in an embedded
environment. The main objective of the implementation is performance evaluation.
The extended access control is expected to have performance overhead, because it considers the additional sequential information. We measure the performance overhead of the implementation result, and compare it with the one of other systems. Additionally, the design structure of the implementation is briefly presented, too.

Consequently, the advanced concept of access control proposed in this dissertation can be applied to development of trusted operating systems to protect the systems from various intrusions. The Coloured Petri Net based access control model can be used to test and verify security-related properties of a system. In addition, the simple embedded security kernel example in this thesis shows the guideline of the implementation of the proposed access control scheme.
안전한 운영체제(Trusted Operating System, TOS)는 기존의 정보보호 소프트웨어들에게서 발생하는 문제점을 해결하고, 보다 원전적인 해법을 제공하기 위하여 제안되었다. 현재, 침입탐지시스템(Intrusion Detection System, IDS)이나 방화벽(firewall) 등의 제품들이 정보보안을 목적으로 적용되고 있다. 그러나, 이들은 컴퓨터 시스템의 구조적 특성이에서 비롯되는 약점을 내재하고 있다. 침입탐지시스템, 방화벽 등은 운영체제가 제공하는 서비스를 기반으로 시스템의 응용 레벨(application level)에서 실행되는 소프트웨어들이나, 따라서, 같은 응용 레벨에 존재하는 공격 프로세스가 이들을 우회하거나, 이들의 보안 메커니즘을 무력화할 수 있다. 현재 공개되어 있는 많은 우회 기법과 공격 사례들이 이러한 취약점을 실증하고 있다.

그러나, 안전한 운영체제는 응용 레벨 보다 아래쪽에 위치한 운영체제 커널 레벨(kernel level)에서 보안 메커니즘을 제공한다. 그러므로, 이보다 상부에서 동작하는 프로세스들이 보안 메커니즘을 우회할 가능성을 원천적으로 배제한다. 또한, 커널 레벨의 연산들은 응용 레벨의 연산들에 비하여 직접적이고 단순한 구조로 이루어져 있으므로, 보다 정교 효율적이고 분석에 용이하다. 따라서, 커널 레벨에서 보안 메커니즘을 제공하는 안전한 운영체제는 기존의 정보보호 제품들에 비하여 건고하고 원천적이며, 효율적인 보안 메커니즘을 제공한다.

안전한 운영체제는 신뢰할 수 있는 전산 환경(Trusted Computing Base, TCB)의 구현이며, 기존의 운영체제 서비스에 보안 커널을 이식하는 형식으로 구성된다. 안전한 운영체제의 동작 기반을 제공하는 보안 커널(security kernel)은 참조모니터(reference monitor)의 구현이며, 참조모니터의 접근통제 서비스는 안전한 운영체제의 기반을 이룬다.

그러나, 현재 보안커널 내 접근통제의 기능에는 한계가 존재한다. 안전한 운영체제의 보안 메커니즘 만으로는 다양한 형태의 공격으로부터 시스템을 완벽히 보호할 수 없는 문제점이 존재한다. 이러한 문제점을 극복하기 위해서는 보안 커널이 공격의 형태에 따라 적절한 대응을 해야 한다. 이러한 대응은 보안 커널이 동작하는 동안 계속적으로 수행되어야 하며, 이 과정에서 보안 커널의 동작 기반을 제공하는 보안 커널의 동작 기반은 안전한 운영체제의 가장 중요한 요소중 하나이다.

안전한 운영체제의 보안 메커니즘은 다양한 형태의 공격으로부터 시스템을 완벽하게 보호함으로써, 시스템의 안전성과 보안성을 강화할 수 있다. 이러한 보안 메커니즘은 시스템의 보안성과 안전성을 보장하고, 시스템의 사용성과 복구성을 확보하며, 시스템의 이용성을 높일 수 있다.
수 없다. 몇 몇 공격들은 접근통제의 보안커널 위에서 아무런 제약없이 성공적으로 수행된다. 이러한 보안 운영체제의 기능적 한계는, 현존하는 접근통제 기법들이 공통으로 내재하고 있는 기능적 한계로부터 비롯된 것이다. 본 논문은 기존 접근통제 기법이 가진 문제점에 대하여 지적하고, 이를 해결하기 위한 방안을 제시한다. 해결 방안의 하나는 접근통제의 개념을 확장하여 기능적 보안을 되는 것이다. 이와 관련하여, 본 논문에서는 다음과 같은 연구 결과를 제시한다.

첫째, 안전한 운영체제 개발을 위해 역할기반접근통제(Role Based Access Control, RBAC)를 확장한 접근통제 기법을 제안한다. 접근통제의 확장을 위하여 연산의 실행 순서를 검사하고, 부정적 권한(negative permission)의 개념을 도입한다. 확장된 접근통제는 접근제계의 집합에 순서(order)를 부여하여 절차(procedure)의 형태로 정의한다. 또한, 특정 절차들을 시스템에 의해 행위로 규정하기 위하여 부정적 의미를 부여한다. 이를 통해, 시스템은 전통적 의미의 접근통제 정보 및 순서화된 연산 정보에 기반, 접근의 합법성을 판단할 수 있으며, 특정 연산 수행 절차를 공격으로 인식하고 거부할 수 있다.

둘째, 확장된 접근통제 기법을 위한 정형 모델을 제시한다. 정형 모델을 이용한 시스템 명세는 접근통제 기법의 구현을 위한 단계적 절차이며, 정형 검증은 시스템 운영시, 보안 설정(security configuration)이 올바른지의 여부를 검사하기 위하여 필요하다. 시스템의 보안 설정은 시스템의 안전에 결정적인 영향을 주지만, 보안 설정에 내재된 오류나 결점을 사람의 손으로 찾아내기는 어렵다. 그러나, 정형기법은 이를 자동화할 수 있도록 돕는다. 또한, 정형기법을 통해 검증한 사실은 수학적으로 항상 정리임이 증명된다.

접근통제 기법들이 다양한 정형기법을 통해 모델링 되어왔다. 기존의 역할기반 접근통제 또한 집합 기호에 기반하여 명세(specification)되어왔다. 그러나, 이들 명세기법은 종류로 제안한 확장된 접근통제 기법의 순서화된 연산집합을 기술하기에는 부적절하다. 그러므로, 본 논문은 컬러드 페트리 넷(Coloured Petri Net) 정형 기법을 이용하여 확장된 접근통제기법의 정형 모델을 제시한다. 컬러드 페트리 넷은 형(Type) 정의 및 수식(expression)을 지원하는 상태 기계(state machine) 기반 기법으로, 기존의 접근통제 정보와 확장된 접근통제의 순서화된 연산을 동시에 표현할 수 있어 확장된 접근통제 기법의 정형모델로 적합하다. 시스템을 제시한 정형모델로 표현한 후에는, 컬러드 페트리 넷 자동화도구를 이용하여 보안관련 속성을 검사할 수 있다. 자동화 도구를 이용한 검사는 시스템의 보안 관리를 돕는다.

셋째, 제안된 접근통제 기법의 간단한 구현 결과를 보인다. 본 논문에서 제안한
접근통제 기법은 임베디드(embedded) 환경에서 구현되었다. 구현 목적의 하나는 제안된 기법이 구현되었을 때의 시스템 성능을 평가하는 것이다. 제안된 기법은 기존 접근통제가 사용하는 접근 통제 정보에, 절차적 정보를 추가로 고려하므로 성능 부하(performance overhead)가 발생할 수 있다. 본 논문에서는 제안한 접근통제 기법을 구현하여 성능을 측정하고 이를 다른 시스템과 비교한 결과를 보인다. 또한, 제안기법의 구현을 위한 시스템 디자인을 간단히 제시한다.

본 논문에서 제안한 보다 진보적인 개념의 접근통제 기법은 안전한 운영체제의 보안 커널 구현에 사용될 수 있다. 이를 통해, 기존 접근통제로는 불가능했던 다양한 공격으로의 보호가 가능하다. 또한, 제안한 접근통제 기법을 위해 제안한 컬러드 페트리넷 기반 정형모델을 이용하여 시스템을 명세하고, 보안 관련 속성을 검사하는 것이 가능하다. 컬러드 페트리넷 정형 기법 및 자동화 도구의 지원은 보안관리에 도움을 주며, 결과적으로 시스템의 안전성 유지에 기여된다. 이와 더불어, 논문에서 제시한 임베디드 환경에서의 간단한 구현 결과는 제안한 기법의 구현을 위한 사례로 사용할 수 있다.
Acknowledgements
## Contents

Abstract (English) i  
Abstract (Korean) iv  
Acknowledgements vii  
List of Contents ix  
List of Figures xi  

1 Introduction 1  
   1.1 Introduction to the Trusted Operating Systems ............... 1  
   1.2 Thesis Overview and Organization ............................. 6  

2 Access Control Issues in Trusted Operating Systems 9  
   2.1 Introduction .............................................. 9  
   2.2 Current Access Control Schemes of Trusted Operating Systems . . . . 9  
   2.2.1 Discretionary Access Control and Trusted Operating Systems . 13  
   2.2.2 Mandatory Access Control and Trusted Operating Systems . . . 18  
   2.2.3 Role Based Access Control in Operating Systems ........ 23  
   2.3 Motivation of the Extended Role Based Access Control ........... 28  
   2.4 The Approach of This Thesis .................................. 32  
   2.5 Summary .................................................. 34  

3 The Extended Role Based Access Controls 37  
   3.1 Introduction ................................................ 37  
   3.2 Related Work ................................................ 37  
   3.2.1 The History based SOD Enforcements ....................... 38  
   3.2.2 The IDS Systems ........................................ 40  
   3.3 The Abstraction in Role Based Access Control .................. 42  
   3.3.1 Extended Properties and The Separated Abstraction ............ 44  
   3.3.2 The Other Reasons of The Separated Abstraction ............. 46  
   3.4 Extended Role-Based Access Control Model ...................... 49  
   3.4.1 Core E-RBAC Model ...................................... 50
3.4.2 Constrained E-RBAC Model ........................................ 54
3.5 Additions to E-RBAC .............................................. 59
  3.5.1 Role Hierarchy ............................................ 59
  3.5.2 Test of the Relations ..................................... 61
3.6 Analysis .......................................................... 69
3.7 Summary .......................................................... 71

4 Coloured Petri Net Formal Model for E-RBAC .................. 72
  4.1 Introduction ................................................... 72
  4.2 Related Work .................................................. 77
    4.2.1 Access Control Models .................................. 78
    4.2.2 Alloy specification of the core E-RBAC model ....... 86
    4.2.3 Petri Net Models ....................................... 90
  4.3 Constrained Coloured Petri Net ............................... 95
    4.3.1 Basic Concepts ......................................... 96
    4.3.2 Constrained Coloured Petri Net ....................... 97
  4.4 Analysis ....................................................... 103
    4.4.1 An Example Configuration ................................ 104
    4.4.2 Analysis by Simulation .................................. 107
    4.4.3 Analysis by Verification ............................... 110
  4.5 Summary ....................................................... 112

5 Implementation and Performance Evaluation ..................... 114
  5.1 Introduction ................................................... 114
  5.2 Implementation ............................................... 115
  5.3 Performance Evaluation ....................................... 117
  5.4 The verification approach for implementations .......... 119
  5.5 Summary ....................................................... 128

6 Conclusions ....................................................... 129

References .......................................................... 131
List of Figures

1.1 The layers in a computer system. .................................. 2
1.2 Mistake of a silly detective. ....................................... 5

2.1 Permissions are defined as the combination of operations and objects. 10
2.2 An access control matrix. .......................................... 14
2.3 The access control matrix of Fig. 2.2 and its alternative data structures;
   (a) Capability Lists and (b) Access Control Lists. ............... 15
2.4 An example multi-level hierarchy. .................................. 21
2.5 The roles defined in a branch office of a bank. .................. 25
2.6 Role hierarchy which is originated from user organization[63]. .... 27
2.7 A role hierarchy having the roles in SOD relations. ............. 28
2.8 Access controls. ..................................................... 29
2.9 The discrete decision procedure of the current access controls. . . . 30
2.10 The example of a race-condition attack. ......................... 31

3.1 A role hierarchy in a branch office of a bank. .................. 46
3.2 The modified role hierarchy. ....................................... 47
3.3 A created cycle in role hierarchy. .................................. 47
3.4 Extended RBAC entities and relations. ........................... 50
3.5 Extended RBAC entities and relations with procedural restrictions. . 55
3.6 Testing procedure of an access with procedural restrictions. ...... 58
3.7 A simple role hierarchy. ............................................ 59
3.8 The roles that are not separated. ..................................... 61
3.9 A redundant relationship. ............................................. 63
3.10 Relationship of E-RBAC entities. ................................. 64
3.11 An example of the ER-graph. ........................................ 66
3.12 Example definitions of positive/negative procedure. ......... 69

4.1 A summarized chart of the Trusted Computer System Evaluation Criteria .................................................... 74
4.2 The conceptual diagram of RBAC concept. ....................... 82
4.3 The diagram of the constrained E-RBAC Model. ................ 89
4.4 Petri Net and Coloured Petri Net graphs that express a same system. ......................................................... 93
4.5 A basic components of the CCPN model. ......................... 97
4.6 A CPN modeling result (1): A system with PC examples. .... 104
4.7 A CPN modeling result (2): The positive PC example. ....... 105
4.8 A CPN modeling result (3): The negative PC example. ....... 106
4.9 The simulation results of the positive procedural constraints which is related to the log file manipulation. The three possible execution sequences of (a) open-read-close ........................................ 107
4.10 (b) open-write-close ................................................. 108
4.11 (c) open-close ...................................................... 108
4.12 The simulation results of the negative procedural constraints which is derived from the sendmail daemon race condition attack. The detected and isolated attack trial is shown in (a) the negative PC subnet .... 109

- xii -
4.13 (b) overall diagram ............................................. 110
4.14 The Result of Liveness Check of The Negative PC. .............. 111
4.15 The Result of Liveness Check of The Negative PC with The Modified
   Configuration. .................................................. 111

5.1 Examples of an attack execution and its detection. .................. 117
5.2 Comparison of the execution time between the original kernel and the
   modified kernel using E-RBAC from (a) a simple execution program, (b)
   a copy program, and (c) a log read program. ........................ 118
5.3 The overheads ................................................... 118
5.4 The comparison of the system call approach and the LSM approach. . 122
5.5 The founded vulnerability in LSM implementation. ................. 126
5.6 The example of the misplacement of authorization. ................. 127
Chapter 1

Introduction

In this chapter, we briefly introduce overall scope of the dissertation. This thesis is motivated from the design of existing trusted operating systems. We propose an extension of access control and its formal model, and they are discussed based on the context of trusted operating systems. In addition, they are applied to an embedded operating system to show an implementation example. Therefore, the concept of trusted operating system is first introduced, and then the contents of this thesis are outlined.

1.1 Introduction to the Trusted Operating Systems

The trusted operating systems are the security-enhanced operating systems. The term of trusted operating system has been evolved since 1980s. The trusted operating systems classify the stored information and provide security mechanisms for keeping secrecy, integrity and availability of the information. The National Security Agency (NSA) of the United States has evaluated the systems since 1984[1].

The trusted operating systems aim to solve the problems resulted from the weakness of application-level security solutions. In these days, threats to the computer systems are being more serious. Hence, several security solutions such as intrusion detection systems and firewalls have been adopted into business, educational, and private environments to strengthen security of the computer systems. They are installed and
operated on many systems and provide security mechanisms that screen false information, unauthorized accesses, intrusive trials, and so forth.

However, the techniques bypassing security mechanisms in those security solutions have also evolved[2, 3]. As a matter of fact, it may be the contradiction of impenetrable shield and piercing-all spear. Various intrusion techniques have appeared following the introduction of security applications to computer and network systems. Hence, we have worried about possibility of bypassing and we have patched the security applications repeatedly after the installation.

The weakness of current security solutions is originated from their structural positions in computer systems. The structure of computer systems can be simplified as shown in Fig. 1.1. Operating systems provide such services as system calls, commands, and application interface calls to the application software. The vulnerable solutions provide security mechanisms at the application-level based on the services from the operating system layer. However, dangerous processes can be also programmed and executed in the application level. Then, it is not difficult to bypass the application-level security mechanisms using the same services that the operating system provides. To evade the watchdog, attack programs use API functions or kernel services. It is
hidden or replaced with the other legal one that their signature information such as
the function pointers or addresses are stored in main memory.

Even though the application-level security solutions are executed with more privi-
lege than user processes, they are not located at the bottleneck point where they can
observe and mediate all calls of user processes. Consequently, their security mech-
anisms are not strong enough to enforce unavoidable controls. To provide stronger
security mechanism, they should be located at the lower level than applications.

Private firewalls and host-based intrusion detection systems are the host-based se-
curity solutions. They are installed on any computer system and protect the system
from various intrusions. They have the security defect that was previously discussed.
It is not a problem of the host-based security solutions only. Network-based security
solutions are also installed on any computer or embedded in network devices and they
watch the interchanged information. Such network-based solutions as network-based
intrusion detection system and firewalls are also vulnerable to the security threats.
Packet watching solutions are incapable of disguising packets and collaborating in-
ternal hosts. It can be also broken down the computer system where network-based
solution is operated. If they cannot protect the intranet hosts, the network-based
solutions are surely regarded as being useless.

The researches seeking more trusted systems are started from the weaknesses of
application-level security solutions. To solve the security problems more fundamen-
tally, the trusted operating systems should provide security mechanisms at the kernel-
level that is lower than the application-level. The security controls must be enforced
as a part of an operating system, and then provide unavoidable control processes from the viewpoint of application-level processes. Loscocco et al. [5] pointed out the security threats to modern computing environment could not be solved without trusted operating systems. Additionally, he emphasized the essence and importance of trusted operating systems with a proverb: all security effort without trusted operating systems would result in “Fortress built upon sand[4]”.

At the center of trusted operating system, there is a security kernel. Gollman [6] described the concept of security kernel in the terms of the Trusted Computer System Evaluation Criteria [7] that have been used as the evaluation guideline of trusted operating systems. The related terms can be summarized as follows.

- The **trusted operating system** is the implementation of Trusted Computing Base.

- The **trusted computing base** is the totality of computer system’s protection mechanisms and it enforces a unified security policy to guarantee confidentiality, integrity, and availability of the system [6, 7, 8].

- The **security policy** is the rules and practices describing how sensitive information is processed, delivered, and protected [9].

- The **reference monitor** is a conceptual model of access controls that mediates all accesses in a system.

- The **security kernel** is the software and hardware elements that enforce access control. The security kernel mediates all accesses of a system, and then only
In a word, the security kernel is the core of trusted operating system and they are the implementation of reference monitor and trusted computing base, respectively.

<table>
<thead>
<tr>
<th>Conceptual level</th>
<th>Implementation level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference monitoring</td>
<td>Access control</td>
</tr>
<tr>
<td>Reference monitor</td>
<td>Security kernel</td>
</tr>
<tr>
<td>Trusted computing base</td>
<td>Trusted operating system</td>
</tr>
</tbody>
</table>

The main security service of trusted operating system is the access control of se-
curity kernel. However, current access control is still limited in its function. The limitation originates from the discreteness of access control information. Whenever an access is done, a security kernel gathers security-related information from the access subject, access object, and the system environment. The information is used for an access decision, but disused after the decision. In the decision making process, associated or hidden information between accesses is not considered. Consequently, a set of operations can be allowed separately though the execution of the set results in an attack. In other words, if an attack is composed of ordinary operations so that the operation does not violate the access rules and the security policy of system, then the access control system does not deny the attack. This brings a functional limitation in access controls. The limitation is similar to the mistake of a silly detective who estimates direction of a thief by confirming with only one of many footprints (See Fig. 1.2).

1.2 Thesis Overview and Organization

In this thesis, we discuss the concept and model of extended access control preventing the exploitations that consist of ordinary operations. For the extension, we adopt the notion of behavior modeling which have been applied to the intrusion detection systems(IDS). By the extended control, access decisions are made not only from instantly gathered information, but also associated information of accesses. The associated information is stored as a form of procedure. Moreover, we adopt the concept of negative permissions to describe attack sequences.

Finally, the access control system tests whether an execution of each operation
completes an attack procedure, and then blocks intrusions effectively. The integrated security service of intrusion detection and access control is suitable for embedded systems. In embedded environment, we do not have enough resources to operate such a security solution as intrusion detection system.

The contents of remaining chapters are summarized as follows.

In Chapter 2, current access control schemes of trusted operating systems are introduced. In addition, the weakness of current access control is pointed out. The weakness is originated from insufficient functionality of access control schemes, and it brings security problems into the systems where the access control is the main security mechanism. In this chapter, the approach to solve the security problems is briefly introduced.

In Chapter 3, the concept of extended role based access control is presented. The Extended role based access control extends and gives additional constraints to the permission abstraction of previous role based access control. The additional constraints are given to express execution sequences of system operations. Considering the execution sequence in access control mechanism, we can supplement insufficiency of current access control schemes. Finally, it extends the functionality of access controls.

In Chapter 4, a formal model of extended role based access control is discussed. The previous access control models are not enough to express the execution sequences as well as access matrix information. Therefore, a formal model is suggested based on the Coloured Petri Net formalism. With the Coloured Petri Net model, we can express both access relation information and execution sequence information.
Moreover, it is possible to test security-related properties with automatic simulation and verification tools.

In Chapter 5, an implementation example of extended access control is presented. We implemented the extended role based access control concept and its formal model in an embedded operating system. The aim of implementation is to show the performance overhead of the extended access control. The extended access control scheme is expected to have some performance overhead due to the consideration of execution sequences. We evaluate the performance overhead of embedded operating system implementation.

Finally, we conclude this paper in Chapter 6. This dissertation is summarized in this chapter. In addition, next research issues are briefly listed with respect to access control and trusted embedded operating systems.
Chapter 2

Access Control Issues in Trusted Operating Systems

2.1 Introduction

Security kernels are the main part of trusted operating systems, and access controls are the important function of the security kernels. Without support of the security kernels, security policies cannot be enforced in trusted operating systems. Therefore, it is important for the system security that access control schemes provided in security kernels. The access control schemes characterize security services of trusted operating systems.

In this chapter, we briefly introduce the access control access control schemes that have been adopted into the development of trusted operating systems. Next, the common weakness of the access control is pointed out. That is the motivation of this thesis. Then, the rough idea of overcoming the weakness is introduced.

2.2 Current Access Control Schemes of Trusted Operating Systems

At the basis of the security services of the trusted operating systems, there is access control. The access control is the process limiting accesses between access subjects and objects[10] by concerning authority of the subject and classification of the object.
Figure 2.1: Permissions are defined as the combination of operations and objects.

The access subjects are basically users, the human beings. However, the concept of the access subject can include not only human beings, but also software agents, machines, network elements, and so on. They access and use system resources to achieve their purposes. For the simplicity, we limit the boundary of the concept as users and their processes. In a computer system, users’ authorities are delegated to their processes, and the processes act on behalf of the users. In the same manner, the other non-human subjects should be substantiated as processes of a system, and then the processes carry out their tasks.

The access objects are originally the system resource objects. There are system resources such as memory spaces, data storages, network sockets, and so on. Several operations are provided for a resource object in a computer system. For example, for a file object, various operations are provided such as read, write, append, link, unlink, create, delete, open, close, and so on. An access is differently understood with the operation used for the access. Although a subject accesses to a same system resource several times, all accesses have different semantics if the subject performs different operation at each moment. Therefore, in the viewpoint of access controls, the access objects are interpreted as the operations on the resources, not just the
resources themselves. Based on this idea, the permissions of the standard RBAC model[11] are derived from the operations and objects (See Fig. 2.1). A permission is the authorization to execute an operation on an object. Permissions have been defined differently in access control models and implementations.

On the other hand, in some access control model, the access targets are the combinations of access modes and resource objects[12]. In BLP model [12], a subject accesses an object with an access mode. The model abstracts the possible behaviors of subjects into the a few distinguished actions. As a result, five access modes of read, write, append, execute and control are defined in the BLP model.

In the RSBAC (Rule Set Based Access Control) implementation[15], authorizations are defined in more detail. Permissions are the accesses to objects of certain types in certain access modes.

To build a secure system, security policy is established, specified with a security model, and implemented with effective mechanism [16]. In the same manner, the access control is realized by following steps:

- **List Requirements:** It is necessary to get requirements before building an access control system. The objective of the control, the access subjects and objects, the available conditions of accesses should be listed. Access control system can have different form according to the requirements.

- **Establish Access Control Policy:** Distinguish authorized states and unauthorized states considering the requirements. Access control policy is a statement describing those states. Security policies are differently developed for the requirements.
In commercial environments, security policies are developed to provide integrity.
On the contrary, policies mainly concern confidentiality in military systems. Similarly, access control policies have differently developed for the requirements and target environments.

- **Describe Access Control Model**: Specify the policy based on mathematical model. Access control model describe policies without ambiguity based on the mathematical foundation. Models help to understand polices by giving clear idea of the elements and statements. In addition, they provide clear specification of the functions and mechanisms to be implemented.

- **Implement Access Control Framework**: Implement an access control system that enforces the security policy using appropriate mechanisms.

- **Run Access Control Framework**: Establish configuration for the implemented access control system. Manage and administrate the implemented framework. When there is any change in the environment where the access control system is executed, reflect it to the configuration.

Among the above steps of access control, the most influencing factor of the access control system is the access control policy. Access control policies or access control schemes influence the feature of model, implementation mechanisms, and implemented system. There are three main access control schemes in access controls: *Discretionary Access Control*, *Mandatory Access Control*, and *Role Based Access Control*.

Traditionally, general UNIX-compatible operating systems have controlled accesses
under the discretionary access control policy. While discretionary access control is a flexible scheme and many systems adopt it as their main security policy, it does not support a sufficient level of security. Therefore, early trusted operating system developments adopted a supplementary policy of mandatory access control[17, 18, 19]. Yet, mandatory access control has too strict limitations for commercial applications. Therefore, it has not widely adopted due to the lack of flexibility in executions although it enforces very strong security policy. Hence, recent trusted operating system developments[15, 20, 21] introduce role based access control. Role based access control provides several advantages covering the weak points of previous two schemes.

2.2.1 Discretionary Access Control and Trusted Operating Systems

Discretionary access control (also called as identity based access control) mediates accesses based on the identity information of access subjects. Subjects access objects only when the subjects have permitted identities. The main characteristic of discretionary access control is that the access control is at the discretion of the user. The owner of an object decides who is allowed to access the object. In addition, the owner can change the owner information of the object. In other words, owners can set security-related information of their object. The voluntary decision of the owner is the reason why the access control scheme named as ‘discretionary’. As a result, consistent and system-wide security policy is not enforced in the system. The owners maintain security policies individually.

To enforce the discretionary access control policy, a system maintains the informa-
An access control matrix is generally sparse matrix and it is not easy to be implemented directly. Therefore, two kinds of substitutions are implemented in real systems.

- Capability Lists: A list is called as a *capability* of a subject, and the list describes accessible objects and access rights for the subject. Capability lists are the rows of subjects of the access control matrix. Fig. 2.3 (a) shows the capability list from the previous access control matrix example. By the lists, it is easy to understand what are the permitted objects and rights for a subject. It is easy to create a list for a new user and to delete access rights for a seceded user from the system.

- Access Control Lists: An *access control list* includes subjects and their access
Figure 2.3: The access control matrix of Fig. 2.2 and its alternative data structures; (a) Capability Lists and (b) Access Control Lists.
rights for an object. Access control lists are the columns of the access control matrix. The example access control list from the previous access control matrix is shown in Fig. 2.3 (b). Contrary to the capability lists, the management is easy in terms of objects. In Unix systems, a kind of access control list, called as permission-bit, is attached to each object.

The representative example system of discretionary access control is the traditional Unix System. If a user creates a file, then the user gets ownership of the file and sets permission of others. The basic information is a 9-tuple of bits, and each bit represents read, write, execution permissions of the owner, users who are in same group with the owner, and other users (generally, the initial permission is set automatically following ‘umask’ information). The permission-bit is attached to each resource objects, and used as the basis of access mediation of the system. The owner can change the permission-bits of their own resources. It is a flexible scheme. The accesses are permitted or not by users’ decisions. The users change the bits based on the present system context. For that reason, most systems are operated under the discretionary policy currently.

On the other hand, the discretionary policy does not provide enough security in some cases. For example, a user produces a file containing valuable information. When the user gives the authority of referring the information to a particular group of users, the user may expect that the information in the file is kept between the members of the group and the other users out of group cannot read the file. In discretionary access control, it is not guaranteed that kind of a limitation. Someone in the group can produce a carbon copy of the information if he can read the information. After
the creation of the copy, he owns the copied information, and then he can transfer the right of referring the information to anyone. As a result, the original creator of the information cannot control the rights on the information. In other words, the flow of information is not controlled in the discretionary policy.

Therefore, even if we can say the current state of the system is perfect from the viewpoint of security, we cannot say the next state is also secure. There are so many administrators intervening security of the system where discretionary access control is enforced. Stemming from that, a discretionary access control system is vulnerable to the attack from ‘Trojan horse’ programs. If a Trojan horse process gets an authority for secret information, we cannot prohibit the malicious process from carrying the information out.

Also, discretionary access control totally depends on identities with no concern of the semantics of information. Therefore, if a Trojan horse masquerades as a legitimated user, the system cannot prohibit important information from leaking.

Besides, the discretionary policy often comes into conflict with the resource management policy of enterprise systems. In enterprise environments, information, which is produced by an employee, is treated as an asset of the organization. Although a user create a file by him, the file should belong to the company. In other words, the owner of the file is the company but the user. The user should be prohibited from delegating the ownership and the permissions on the file. This requirement in enterprise environments is contradictory to the discretionary policy, because the producer of information owns the information and manages the permissions in the discretionary policy.
Currently, trusted operating systems do not enforce the discretionary policy only, but they enforces supplementary access control scheme additionally.

2.2.2 Mandatory Access Control and Trusted Operating Systems

In contrast to the discretionary scheme, a mandatory access control system enforces a system-wide security policy. There are unconditional access rules are applied, and all interactions between subjects and objects should follow the rules. Mandatory access control (also called as rule-based access control) mediates all accesses based on the constant rules. Moreover, users cannot alter the security-related information of a mandatory access control system. Even if a user produces information, the user cannot change the security label of the information. The user cannot delegate the own authority to others. Instead, the security officer is in charge of setting, coordinating, and managing security-related information of a system.

Mandatory access control mediates accesses also based on the labels; Clearances are given to each access subject and the classifications are given to each access object. All operations in a system are controlled by the label comparison. The operations are permitted or not by the result of the comparison.

The two most famous mandatory access control models are Bell-LaPadula model[12] and Biba model[13]. The models define the dominance relation between entities to classify the relationships between them. If the security level of one entity is evaluated as higher than or equal to the other entity, then the former entity ‘dominates’ the latter entity.
• Bell-LaPadula (BLP) Model: In BLP model, three main security properties are defined. The system only allows accesses if the subjects and objects keep the simple properties.

– ss-property (simple security property): A state of the system satisfies the ss-property, if for each access of which the access operation is read or write, then the security level of the subject dominates the classification level of the object. This characteristic is called as no read-up security policy.

– *-property: A state of the system satisfies the *-property, if for each access of which the access operation is append or write, then the security level of the subject is dominated by the classification level of the object. This is called as no write-down security policy. In addition, if there is another access from the same subject to the other object and the access operation is read or write, then former object should dominate the level of the other object.

– ds-property: A state of the system satisfies the ds-property, if for each access from a subject to an object, the access mode is defined in the access matrix entry of the subject and the object. This property corresponds to the access rule of the discretionary policy.

In other words, a subject can read an object if the subject dominates the object. A subject can write information to an object if the object dominates the subject. By these constraints, information always flows unidirectionally into the higher
levels. This model guarantees confidentiality for systems in which the secrecy is the most important such as military systems. However, it is not provided that integrity of information in higher level. The low level subject can alter the information of the higher-level object following the principle.

• Biba Model: In Biba model, a subject can write to the lower level object. On the contrary to the BLP. Read operations are allowed from the lower level. Therefore, upper level information cannot be altered by the lower level subjects. It provides integrity for higher-level objects in a system.

BLP and Biba models order subjects and objects vertically. For example, all documents are labeled as ‘TOP SECRET > SECRET > CONFIDENTIAL > UNCLASSIFIED’ in military.

On the other hand, it is possible to classify subjects and objects into groups horizontally. Compartment based policy classifies all entities into compartments. For example, a storage service center providing servers and mass storages to commercial companies should divide their resources into the sections of companies. It is important to keep the privacy of each company. Hence the sectors of the storages can be categorized as SAMSUNG, LG, HYUNDAI, and so on. Subjects’ accesses are controlled based on the compartment information of the subjects by this policy.

Multi-level policy control access based on vertically ordered- and horizontally separated-labels of subjects and objects. Therefore, multi-level policy can be applied to an environment as enterprises where entities are ordered and classified in organization hierarchies. For example, there are departments such as the general affairs, the business,
and the accountants’ in a company. They may have positions such as a chief, a second chief, members that ordered by their authorities in the organizational hierarchy of the company. In the multi-level policy, the authority of two subjects is different if they belong to different compartment, even though the subjects have same level.

Because the multi-level policy considers compartment information as well as the level information, the dominance relation is newly defined expanding the dominance relation of BLP. For the definition of the dominance relation, assume that a system has labels as follows (Fig. 2.4).

- A hierarchy: TS > S > C > U
- A set of compartments: A, B, C, D

In the system, an entity has the label indicating the rank in the hierarchy, and also the set of label of compartments in which the entity belongs.

The extended dominance relation $\leq_d$ is defined as follows:

**Definition 2.2.1 (Dominance relation, $\leq_d$):**
For the dominance relation \( \leq_d \), \( a \leq_d b \) is true between two entities \( a \) and \( b \) if

- the level of \( b \) is higher than- or equal to the level of \( a \), and

- the compartments of \( b \) include the compartments of \( a \). That is, \( a.compartments \subseteq b.compartments \).

The multi-level policy can be modeled mathematically with a lattice structure. Therefore, the characteristics of the multi-level policy consistent with the mathematical characteristics of the lattice and the properties of the system can be investigated mathematically.

Being compared with the discretionary policy, the most important characteristic of the mandatory policy is that security-related information is administrated in a centralized manner. Contrary to the discretionary policy, mandatory policy does not let users modify the security-related information of the object or delegate the permissions. All accesses of a system should follow the security rules of the system without an exception. The system classifies subjects and objects, gives them security labels, and defines the unconditional security rules. Based on these strong limitations, a mandatory access control system is strong against Trojan horse attacks. Information and permissions are controlled not to flow into unauthorized entities. Therefore, Trojan horses cannot get secret information and they cannot leak out secret information even if they got the information.

Early trusted operating system implementations noticed the advantages of the mandatory policy and have been adopted the policy to make up the insufficiency of
the discretionary policy of the traditional systems. As a result, the mandatory policy is enforced in several current trusted operating system implementations[17, 18, 19].

However, the mandatory policy is so restrictive condition for commercial environments. For example, the chief officer of a company cannot send an e-mail to his team members because they may be ranked as the lower level than his level. The public bulletin board will not work in the multi-level system also by the same reason. Mandatory access control is rather suitable for military applications than commercial environments. In military systems, all subjects and objects can be forcefully classified in security levels.

### 2.2.3 Role Based Access Control in Operating Systems

In 1970s, Role Based Access Control (RBAC) is proposed to satisfy the demands of security processing in multi-user and multi-application environments of companies and public institutions.

Roles are intermediaries lying in the middle of access subjects and objects. They are defined between users and permissions mainly reflecting the organization structure of a particular system such as a company, an institute, and so on. RBAC is the method that controls accesses based on user’s role which is defined in organization. A user can access an object only when the user has roles which are permitted to access the object. A role can be regarded as a virtual entity representing a set of users, and it is useful to authority management.

Roles, the instantiated relations of subjects and objects, are defined by security ad-
ministration. The security officer takes in charge of security administration. RBAC is the centralized method like the mandatory policy from the viewpoint of the policy enforcement. However, RBAC is different from the mandatory policy. There are no such permanent rules as the principles of mandatory policies. The security administrator can regulate rules based on situations. Therefore, the burden of security administration is concentrated on the security officer who is responsible for the security of the system.

RBAC overcome the burdens of security administration by providing conveniences based on the notion of abstraction. The concept of abstraction originated from the abstracted role entities. One important work of the security administration is to define mappings between access subjects and access objects for access control. As the relation information of subjects and objects, roles are re-used after they are defined. The security officer can give or deprive user’s permission by assigning him to a role or withdrawing the assignment. The officer is able to permit or get rid of permission also by associating the permission with a role. There is no need to make direct relationships between users and permissions.

Without roles, the security officer performs annoying work when there is a small change in the system configuration. For example, if a new user enrolls to a system, the security officer specifies the authority of the user and associates him with all necessary permissions. However, it is very difficult to grasp whole subject-object relation correctly. In a multi-user environment like an enterprise, it is more troublesome. Especially, if a manager of a company takes in charge of the security administration, and the manager is not familiar with up-to-date-technology of the system, then the
administrative burdens get into trouble. Usually, the managers of companies are not accustomed to the new technologies[22].

RBAC provides conveniences in such environments that have huge and fluctuant users such as companies or public institutions. RBAC reduces the burdens of the security administration. Security officers just define roles and assign users to proper roles considering users’ job functions, positions, and authorities in an organization[11]. For instance, roles in a branch office of a bank can be defined with relevant authorities

Figure 2.5: The roles defined in a branch office of a bank.
as shown in Fig. 2.5.

- Branch Manager: For all customers, modify their account information, read their accounts, and create or delete an account. Manage all employee information. Administrate the vault of the bank.

- Account Manager: Manage customer accounts. Administrate the vault of the bank.

- Teller: Modify a customer’s information and defray limited amount of money.

- Customer: Read the personal account.

- Auditor: Read all data. Visit and see the vault.

Moreover, roles can constitute a systematic structure, called as hierarchy. Semantically or functionally related roles are associated each other with inheritance relationships. The hierarchy improves reusability of role definitions and it reflects users’ organizational structure that is defined in real world. Finally, the hierarchy reduces administrative costs. Moffett[29] categorized constructing methods in the hierarchy constitutions as aggregation, generalization, and organization. Fig. 2.6 shows a result of the organization.

In addition, roles are useful to constrain subjects and objects. Separation of duty (SOD) relationships can be defined between roles to prohibit two or more roles being assigned to a particular user. The SOD constraints are categorized as Static SOD (SSOD) and Dynamic SOD (DSOD). The SSOD excludes the possibility for a user to
have two roles at the same time. If a system authorized the user to have a role and the other role which is in SSOD relation with the former role, then the authorization conflicts with the SSOD constraints.

Different from SSOD, the DSOD enables a user to activate both roles which are in DSOD relation. He may be authorized to have both roles, but he can activate one of the roles. It is not permitted to acquire both roles at the same time. The example of SSOD and DSOD is shown in Fig. 2.7. In the example, who are authorized to be ‘Branch Manager’ never be able to act as ‘Auditor’ in the system. A ‘Teller’ can be a ‘Customer’ of the bank. However, it is prohibited the user from having both roles. Other various conditions such as access time, access domain can be applied to the set of entities easily based on the concept of abstraction. For example, the role of ‘outside duty’ can be activated out of a company’s domain address, or out of the office hours.

Most advantages of from roles of RBAC are similar to the advantages from ab-
Figure 2.7: A role hierarchy having the roles in SOD relations.

straction approaches of object-oriented technology. By providing several conveniences, RBAC compensates for the lack of ability of the traditional policies. Based on the advantages of RBAC, recent trusted operating system implementations[20, 15, 21] have introduced RBAC as their major security policy.

2.3 Motivation of the Extended Role Based Access Control

When a subject accesses an object with an access mode, the access control system mediates the access by deciding the legality of the access based on such information as the clearance level of the subject, classification of the object, permitted access modes on the object, security rules of the system, and so on. Using the terms of the General Framework for Access Control (GFAC)[31], we can simply figure the access control procedures as Fig. 2.8.

Based on the context of access controls, access control information is gathered from the access subject and the access object at the moment of the access, and the
information is used to decide the legality of the access. The access is ended successfully or with failure by the result of the decision. Next, if a following access occurs, access control information is extracted again from the access subject and the access object for judging the legality of this access. That is, after the decision process is over, gathered information loses its validity and thrown away. The Fig. 2.9 illustrates the discrete use of access control information as described above.

This pattern of access control causes a kind of functional limitation as mentioned briefly in the Sect. 4.1. In the process of the access control, hidden information or associated information between accesses are not considered at all. In fact, if there is no consideration of the result of each operation, then their authorizations can be checked in parallel.

Figure 2.8: Access controls.
Figure 2.9: The discrete decision procedure of the current access controls.

Assume that permissions on file accesses are defined as \{fexec, fread, fwrite, flink, funlink\} in a RBAC system. When a user tries to remove a link of the file, a traditional access control system gets information from the user and the file, and checks if the acquired roles of the user have \textit{funlink} on the file. At the next moment, if the user tries to make a symbolic link to the file, a similar process is performed again. However, the correlation between two actions, unlink and link, is not a matter of concern. The access control system considers the latter action is totally independent to the former, so that the permissions are tested severally. If the user have both link and unlink permissions to the file, the user can repeat link and unlink to the file. Though the repetition is an important signature of the race condition attacks, there is no control.

Therefore, that kind of attack is successfully executed in a system consisting or-
The core mechanism of the sendmail race condition attack

```
main(argc, argv){
  for (;;){
    unlink(argv[1]);
    symlink(argv[2], argv[1]);
  }
}
```

The necessary operations for the race condition attack

- Exec: /bin/mail
- unlink
- symlink
- Read SPOOL
- Write SPOOL
- Create SPOOL

The necessary operations for the race condition attack

- Change target object
- `/root/.rhosts`

The exploit code of the sendmail race condition attack

```
racer
main(argc, argv){
  for (;;){
    unlink(argv[1]);
    symlink(argv[2], argv[1]);
  }
}
```

The execution sequence of the operations for the attack

```
Exec: /bin/mail
  unlink
  symlink
  unlink
  symlink
  Write SPOOL
```

Figure 2.10: The example of a race-condition attack.

Ordinary operations. For an example of the attack, one instance of the race-condition attack is depicted in Fig. 2.10.

Race-condition attacks[23] exploiting errors in synchronization of object bindings[24]. The famous race condition attack is the sendmail race condition attack which is introduced by an underground hackers’ group, 8LGM[25]. A particular sendmail daemon writes mail contents to temporary files before deliveries. If an attacker redirect the writing information by replacing the binding to the temporary file with the binding
to an important file, then the attacker write information to the important file with root privilege. The steps of the sendmail race condition attack can be briefly listed as follows; First, send a string such as ‘+ +’ via email with a sendmail program. Second, remove a binding between the mail daemon and a temporary file in the temporary directory. Third, replace the removed binding with a new one making a symbolic link to the important file such as ‘/root/.rhosts’. Finally, the attacker modifies the file with the administrator’s privilege. If the attack is successful, the attacker gains the root privilege.

The operations that are executed for the race condition attack can be listed as shown Fig. 2.10. Each operation of the race condition attack is not special. They are ordinary operations usually permitted to users. However, the execution of the operations in a particular sequence results in an attack.

With the current access control schemes, those attacks are not detected and prohibited. This is the functional limitation of the access control, and it is commonly found in the previous access control schemes. Hence, with the access control scheme, we cannot protect our system from the attacks consisting ordinary operations.

2.4 The Approach of This Thesis

One possible solution of overcoming the limitation is to use associated access information. By using not only instantly gathered access information, but also referring a set of previous access information, we can extend the concept and the function of access control. To extend the view of reference monitoring like that way, we should
associate an access with other accesses and identify the set of accesses in access control policy and model.

In our extension, we group a set of operations, and give relative orders to the operations. By this extension, we can expand the notion of permission to the execution of a procedure from the execution of an operation for an object. A procedure consists of the several partially ordered operations aiming one or more objects.

In addition, we introduce the concept of the negative permission[6]. A negative permission specifies the access operations which a subject is not allowed to perform. Permissions of traditional access control models have been mainly described in the viewpoint of allowed actions. However, we need to describe prohibited behaviors in our extension to find and deny dangerous procedures in a system. Therefore, we clarify the negative actions in our extension.

With the extended access control, we can describe dangerous operation sequences of intrusion, and we can finally describe and recognize harmful executions which have not been detected in the previous access control schemes. For the example of the previous race condition attack, we can refuse the attack trials by prohibiting the repetition of link and unlink after the mail sending operation.

Also, the description of execution sequence gives us additional advantages. We can keep the principle of least privileges more strictly. The principle of least privilege states that access subject should have only essential and legitimated permissions to perform their task.

Assume that we have a set of permissions \{A, B, C, D\}. If they are required for the
task completion of a role, we may assign all the permissions to the role traditionally. However, if the task is completed with the series of the execution as \((A, C, B, D)\), we do not have to allow all combination of permissions to the role. In other words, in the traditional scheme, we have given more authorities to the role than actual needs. If we check sequences of accesses, we are able to reject the abuse of permissions in an unexpected order. The ordered control helps to keep the principle of least privilege.

Also, we can describe the SOD relationships more accurately. If the sequence of \((A, C, B, D)\) is in SOD relation with the permission order of \((B, A, D, C)\), we can observe the principle of separation of duty more properly. The researches on the ordered sequences of operations for SOD enforcement and the approaches on behavior modeling have been dealt with in the field of workflow systems and intrusion detection systems. The relativity and difference will be discussed in Sect. 3.2

2.5 Summary

We have reviewed the current access control schemes in terms of the development of trusted operating systems. Although various access control schemes are adopted into the development of trusted operating systems, it is not sufficient to protect the systems from various attacks because some kind of attacks are executed in the systems with no limit. The insufficiency originates from the processing pattern of the current access control schemes. They do not consider the associated information between accesses. Therefore, we supplement the function of access control by considering the associated access information as well as the simple relation information which derived from the
access control matrix. The associated information is the execution sequence of system operations. Taking the execution sequence as the access control information is adopting the function of intrusion detection into access controls in operating system kernels.

So far, detect and deny of attacks have been out of range of access control. Therefore, there have been no limitations in access control for the attacks consisting ordinary operations especially. Instead, to block those attacks, application-level solutions such as firewalls and intrusion detection systems are adopted into current computing environments. However, those application-level solutions are not fundamental because of the possibility of bypassing as previously mentioned.

Gollmann[6] described the reasons for putting security mechanisms into the lower level of computer systems:

- Higher level of assurance
- Lower performance overheads

For more solid security and performance, we place control mechanisms against attacks at the kernel-level of operating systems. It is the better approach than using current security solution.

There are more reasons to adopt the kind of intrusion detection service to the access control services. As mentioned in the Sect. 4.1, in such environments which have limited resources as embedded systems, it is hard to run intrusion detection systems usually. The application level security solutions generally needs computational power rather than kernel based executions. To apply the trusted operating system technology to embedded systems, we need the integrated and advanced access control service that
is a real-time supporting and low overhead solution based on kernel operations. In addition, if we introduce the intrusion detection mechanism to the security kernel, then we can specify and verify them with a unified model. Verifying the correctness of security kernel is important[6]. In case we design and implement security mechanisms separately, it is hard to examine interactions between independent modules, moreover it is possible to have redundant codes in the viewpoint of the whole system. Modeling and implementing the consolidated model, we can enforce a consistent security policy without redundant or indefinite part between security mechanisms.
Chapter 3

The Extended Role Based Access Controls

3.1 Introduction

In this chapter, we extend the function of access control by considering more information than the current access control schemes. We extend the access control based on the Role Based Access Control (RBAC) model. RBAC provides the concept of abstraction between access subjects and objects. The concept of abstraction helps to give additional properties to the sets of access entities. It is convenient for us to add advanced properties based on the abstraction.

We first introduce the related researches of our extended access control. After the survey, we investigate the characteristics of the abstraction of RBAC, and define the components to express execution sequences.

3.2 Related Work

Related subjects of history based controls have been researched in the field of workflow systems and intrusion detection systems. They are:

- access control schemes supporting separation of duties, and
- history based intrusion detection mechanism.
3.2.1 The History based SOD Enforcements

In workflow systems, the notion of the execution sequence has been adopted to enforce SOD policy: history-based SOD or context sensitive access control.

Sandhu[35] proposed the transaction control expressions to enforce a SOD policy. A transaction consists of multiple steps of roles, and it is required that all participants should be different in the transaction control expression. For the transaction control, historical information is used. When a SOD transaction is executed, the identities of the users are recorded. Using the recorded history, it is possible to prohibit two SOD roles are participate an execution of the transaction.

SOD is the multi-person control policy being defined between plural entities. The targets of SOD can be defined over the heterogeneous network, especially in workflow execution environments. To improve the interoperability in the notion of the transaction control, Papenfus and Botha[36] suggested the SOD description with XML.

TBAC[37] was proposed for dynamic access controls in workflow systems. The execution environment of the system is a distributed and heterogeneous. A workflow task is executed in the defined sequence by the workflow participants at distributed sites. In the dynamic environment, a centralized control can be overhead or cause vulnerabilities. The objective of TBAC is gathering context information for access controls closed to the execution sites. TBAC enforces access controls reflecting the conditions of the tasks’ execution environment just in time. The framework of TBAC also bases on history based scheme of the transaction control[35].

Damian et al.[38] proposed a context sensitive model to keep the principle of SOD
reflecting frequently changing context information in workflow environments. They redefined the concept of the session, and proposed a workflow system in which authorizations are calculated based on the progress of the task execution.

Wainer et al. [39] defined W-RBAC grafting the concept of RBAC on the workflow system. The framework adopted a logic based language, and specifies the conditions for authorization with the language. The framework binds users and roles dynamically, and enforces dynamic SOD.

The previous results have researched for the enforcement of the SOD principles. They adopt the history information for the control, and it is the natural approach because the workflow is defined having the sequence of work items.

The extended access control in this paper is related with the previous work in terms of the expression of operation sequences and delicate SOD enforcement from the sequence control. However, the extension is distinguished from the previous researches in following points.

First, the objective of the extension is different as mentioned above. The related work mainly concerns enforcing the principle of SOD in workflow environments. To the contrary, the main objective of the extension is limiting attacks which are composed of ordinary operations in access control service. Namely, a kind of misuse detection approach is introduced to the previous RBAC model for our trusted operating system development.

The distinguished feature from the different objective is the adoption of the concept
of negative permissions. Without the notion of negative permissions, we have to define all of the authorized operation sequences to deny such an attack as the race condition. In the world of workflow systems, this is not a big problem because workflow tasks are defined having a flow. However, in the trusted operating system development, it is not reasonable to define all of normal accesses with the procedural information. It is analogous to the discussions comparing the anomaly detection approach and the misuse detection approach in IDSs.

Second, the target of separation is different. In the previous researches, SOD relation mostly has been defined in subject-abstractions. SOD aims to exclude the possibility of plural subjects attack a system in conspiracy. In our extension, we bring down the focus into the lower level, permission level. In the extended RBAC model, a combination of some executions is inhibited by separating an abstracted permission with the other. In other words, the constraining relations are defined in object-abstractions. Specifically, it is the \textit{separation of executions}. The traditional SOD schemes have not focused on such fine grain separations. Therefore, they \textbf{cannot deny such an attack as the race condition which is carried out with a single role.}

3.2.2 The IDS Systems

The other research field which is related with the extended access control is the intrusion detection systems, specifically, the behavior modeling approaches.

IDSs are generally categorized as the misuse detection and the anomaly detection. On the other hand, Ko[40] proposed another approach of specification based intrusion
detection. The specification based detection[40] specifies the security-related subset of the insisted behaviors with a formal language, and detects intrusions comparing the specification and audit data. Ko et al.[41] advanced this approach with machine learning methods. The specification approach is analogous to the anomaly detection, hence difficult to be adopted into trusted operating systems; It is hard to specify all of the normal behaviors in OS kernel as previously mentioned.

Another efforts to specify the nominal behaviors and enforce anomaly detection mechanisms are as follows; Sekar et al.[42] specified system call sequences and arguments of valid processes, compiled the specifications to the finite automata. Elbaum and Munson[43] analyzed the internal behaviors with the dynamic software measurement framework based on profiles which are categorized as operational, functional, module, and execution profiles. Büsschkes et al.[44] modeled correct behaviors as transactions, and Ghosh et al.[45] applied machine learning to profile system calls of normal operation condition. With the various specification techniques and profile information, they proposed the anomaly detection mechanisms. It is analogous to the research results in the field of workflow systems (in Sect. 3.2.1), in terms of the modeling of normal execution sequences.

The extended access control in this paper is different from the IDS efforts in following points: the applicable domain and the modeling targets. We paid attention to limiting of the attack which consist ordinary operations, and adopted the misuse detect approach for the extension. Also, the application target of the access control is
the security kernel of trusted operating systems. The access control model of RBAC is extended in the context of the reference monitoring services. Although, the intrusion detection services can be provided in application level as current IDSs are operated, the application level solutions are not fundamental solution as it is discussed in Sect. 4.1. Moreover, we can expect the better performance from the integrated security services based on kernel-level operations. It will be suitable for the embedded environments.

The most important difference of the extended method from IDSs is this: **IDS cannot make a limited boundary of information flows, but our extension does.** The extended method is basically an access control introducing intrusion detection technique by considering not only access matrix information, but also execution sequences as access control information.

From the next section, we build our extended access control concept and model that are distinguished from the previous researches.

### 3.3 The Abstraction in Role Based Access Control

The concept of RBAC is briefly introduced in Sect. 2.2.3. The structure of RBAC consists of three main entities: Users, Roles, Permissions. Users are access subjects and permissions are access objects. The access rules, which are authorized interactions between the subjects and the objects, are instantiated as role entities. In addition, roles are the abstracted entities that give conveniences in security administration because a role represents a set of the users of a set of permissions. In this section, we investigate the characteristics of the entity abstraction to extend RBAC.
The abstractions in RBAC can be classified as subject-abstraction and object-abstraction following the inherent definition of an access; an access is the interaction between a subject and an object. The subject-abstraction is the representation of users as roles based on users’ real-world context. In general, roles reflect the users’ responsibilities and job functions such as ‘Branch Manager’, ‘Account Manager’, and ‘Teller’. It helps security officers to grasp idea of user-to-role assignments. It is easier to assign users to appropriate roles than to associate each of users to permissions directly. Therefore, these kinds of roles are defined as the set of users, and they are assumed as the set of objects in user assignments. Several examples are presented in previous researches[10, 26, 27, 28].

On the other hand, there is the other type of roles. The object-abstraction is the specification of operations or resource objects of a system. As the result of the object-abstraction, we can define the roles such as ‘Accounting’, ‘Auditing’, and ‘Employee Managing’. They consist of primitive operations of a target system, and give ideas of what are the functions of the roles. It gives conveniences in permission-to-role assignments. It is easier to group permissions than to give each of them to users. Contrary to the subject-abstraction, these types of roles are defined as the set of permissions, and regarded as the set of subjects in permission-assignments.

Although the two kinds of abstractions exist together in a single layer of ‘Roles’, RBAC has no component to express the different abstractions. In addition, the abstraction of RBAC has been studied partially and the purpose of the abstraction has been biased in one direction; Most of noted researches on role hierarchy have focused
on the subject-abstractions. Context information for building role hierarchies has been mainly extracted from user organizations. The examples in the researches[10, 11, 29] show how the hierarchies are constructed based on user organizations. Fig. 2.6 presents one of the examples.

However, describing resources as abstracted forms and controlling accesses is important for some systems. Especially, in our extension of access control for trusted operating systems, it is necessary to express the set of operations as an abstracted form of a procedure. The abstraction helps to specify procedures and give additional constraints to the procedures. Hence, we rather concentrate on the specifying the object-abstraction of RBAC than subject-abstraction.

3.3.1 Extended Properties and The Separated Abstraction

For the purpose of controlling accesses with an advanced manner, we add the following properties.

- Procedural information: Order and repetition information are added to the member of a set of operations. By the information, we can constitute a procedure with the set of operations. The order describes the relative execution order of the operation in the procedure. The repetition explains how many times the operation can be executed repeatedly.

- Positive/negative information: The information indicates whether a set of permissions is harmful or not to the system. The main objective of this property is to describe attack processes in our access control model.
For a precise control, other various information can be added also such as access
time, limited lapse from execution start, owner of the objects, and so on. However,
these constraints are not dealt with in this extension for simplicity.

Although the additional information is useful for the object-abstraction, it is not
necessary for the subject-abstraction. For example, roles such as ‘Branch Manager’,
‘Account Manager’ do not need to have the order or the repetition information. Adding
those properties to the subject-abstraction, it rather gives us the following disadvan-
tages.

• Implementation overheads: In an implementation of the abstraction, we provide
data structures and memory storages for the order and repetition information.
In addition, we have to write initializing and handling functions for those fields.
However, the fields and functions do not need to be prepared for such roles as
‘Branch Manager’. It will be overhead for the roles of the subject-abstraction.

• Semantic estrangements: It is not semantically clear that the subject-abstracted
entities and the object-abstracted entities are existing together in a same layer
of ‘Roles’. They have different use and characteristics although they have same
name. Moreover, the coexistence can cause confusions. The coexistence brings
erroneous relationships by introducing cycles in a role hierarchy. It will be dis-
cussed in Sect. 3.3.2 in more detail.

To avoid disadvantages, it is natural to separate the subject-abstraction and the
object-abstraction. Roles for the object-abstraction can be separated from the roles
of RBAC. We name newly separated roles for the object-abstraction as ‘Behaviors’, but keep the name of ‘Roles’ for the subject-abstraction. As a consequence of the separation, the modified concept consists of four main entities of users, roles, behaviors, and permissions.

3.3.2 The Other Reasons of The Separated Abstraction

The abstracted entities of RBAC have different attributes, but they coexist in the same layer of ‘Roles’. The coexistence causes inadequacies of RBAC. Since subjects and objects are mixed up in a single layer, it is possible to have erroneous relationships. The next example shows the case of an erroneous relationship originated from the confusion in security rule establishments.

Fig. 3.1 shows a defined role hierarchy in a branch office of a bank. The directed arrows indicate the relationships between roles; the role on the tail of an arrow inherits
(or includes) the other role on the head side. Each role is defined inheriting other required roles to perform its own task successfully.

Assume that this pre-defined role hierarchy need to be modified for some reason. Sometimes, a branch manager needs to perform an account officer’s task, or an account officer needs to perform a teller’s job. Because a branch manager is an account officer’s superior and an account officer is a teller’s superior, it does not violate security policy as shown in [26, 28]. Then, a manager who is responsible for security administration makes a little change in the role hierarchy.

In the role-hierarchy modification, the following guidelines can be considered.

- Grouping related roles: Categorize and structuring related permissions in groups.

- Using inheritance: If we need a set of permissions, then we inherit a role which have the permissions. It is easy and maximizes the reusability.
We should consider work pattern and criteria of jobs to make inheritance relationships as shown in [34]. Following the above guidelines, a security officer can make the role ‘Account Info Mgmt’ inherit ‘Account Officer’. The ‘Account Info Mgmt’ is one of the included roles of the ‘Branch manager’. Therefore, the security officer builds the inheritance relationship categorizing accounting-related permissions and using the inheritance relationship. In addition, the officer makes ‘Account Officer’ inherit ‘Teller’. Fig. 3.2 shows the result of the hierarchy modification. With these modifications, superiors have authorities of subordinates. That is, ‘Branch Manager’ and ‘Account Officer’ can perform tasks of the subordinates, ‘Account Officer’ and ‘Teller’, respectively.

However, this modification causes an erratic relationship. As a result of the modification in the role hierarchy, a cycle appears in the role hierarchy. Fig. 3.3 shows the generated cycle by the above modifications. The cycle implies that all roles in the cycle are able to have same authorities, and hence the role hierarchy is collapsed. This may result in serious security problem that is violation of the principle of least privilege. For a simple example, a teller can modify his personnel information with the acquired permission of ‘Teller Performance Rating’ which has been originally assigned to ‘Account Officer’. One of the important reasons for the fault in administration is that the user-induced role of ‘Account Officer’ is assigned to the permission-induced roles of ‘Account Info Mgmt’. In consequence, the assignment of user abstractions causes to permission abstractions caused reverse transmission of unexpected authorities and erratic relationships. The situation stems from confusions in role structure. If subject-abstractions and object-abstractions co-exist in a single category of roles, it
is difficult to distinguish both of them and keep the semantic consistency in ordered role-hierarchy correctly.

It is possible to prevent that kind of hazards by means of some administration software that sets and checking hierarchical order [30]. However, it is not the fundamental solution. Better solution is to eliminate the origin of the confusion that is the coexistence of two different attributes in one abstraction layer. In consequence, it is reasonable to divide area of the subjects and objects at the level of the access control concept and model.

In the next section, we extend RBAC so that subjects and objects are abstracted independently. The extension makes the administrator distinguish access subjects and objects clearly.

3.4 Extended Role-Based Access Control Model

We name our extension of the access control as the Extended Role-Based Access Control (E-RBAC). In this chapter, we describe the model of E-RBAC. The description of the E-RBAC model is also extended and modified from the standard RBAC model[11].

The specification of E-RBAC model starts from the core E-RBAC model. Core E-RBAC is the result of the separation of the abstraction from RBAC. The abstraction of RBAC is divided into two abstraction layers, and they are named as ‘Roles’ and ‘Behaviors’. After the core E-RBAC model is described, we introduce the constrained E-RBAC model. Constrained E-RBAC adds the concept of Procedural Restrictions
(PR) to core E-RBAC. PR enables a system to control operations elaborately by restraining behaviors to be executed in order.

### 3.4.1 Core E-RBAC Model

Fig. 3.4 shows the core E-RBAC elements and relations. The basic elements are users, roles, behaviors, and permissions. Users are the subjects of all accesses. Roles are the semantic representatives of users. A role is defined as a set of users. Permissions are objects of all accesses. Permissions are derived from the combinations of resource objects and access modes. Behaviors are the semantic representatives for permissions as roles do. A behavior is defined as a set of permissions. Sessions are the dynamic associations between users and roles. A session describes a mapping between a subject and the roles which are acquired by the subject in runtime. As a result, users have two types of roles; authorized roles and activated (or acquired) roles. Users are permitted to have authorized roles in a system. A user opens a session to perform a particular task in runtime, and a subset of the authorized roles is given to the user during the session.
Basic relations are subject assignments (SA), access rules (AR), and object assignments (OA). As Fig. 3.4 shows, SA consists of one-to-one user-session mapping and many-to-many session-role associations. AR is the relation between roles and behaviors. In terms of reference monitor, roles and behaviors are the representatives of the actual access subjects and objects, respectively. Hence, AR is the essential security rules limiting subjects’ accesses to objects. OA is the relation between behavior and permissions. Above descriptions are formally described as follows.

Definition 3.4.1 (Core E-RBAC) :

- USERS, ROLES, BEHAVS, and PERMS : the set of users, roles, behaviors, and permissions, respectively.

- $SA \subseteq USERS \times ROLES$, the many-to-many user-to-role assignment.

- $assigned\_users : (r : ROLES) \rightarrow 2^{USERS}$, the mapping from a role $r$ onto a set of users,
  
  $\quad - assigned\_users(r) = \{ u \in USERS \mid (u, r) \in SA \}$

- $OA \subseteq BEHAVS \times PERMS$, the many-to-many permission-to-behavior assignment.

- $assigned\_permissions : (b : BEHAVS) \rightarrow 2^{PERMS}$, the mapping of a behavior $b$ onto a set of permissions,
  
  $\quad - assigned\_permissions(b) = \{ p \in PERMS \mid (p, b) \in OA \}$
• **SESSIONS**: the set of sessions.

• **user_session**\( u : \text{ USERS} \) → **SESSIONS**, the mapping from a user \( u \) onto a session.

• **session_roles**\( s : \text{ SESSIONS} \) → **ROLES**, the mapping from a session \( s \) onto a set of roles.

• **AR** ⊆ **ROLES** × **BEHAVS**, the many-to-many role-to-behavior assignment.

• **assigned_behaviors** : \( r : \text{ ROLES} \) → \( 2^{\text{BEHAVS}} \), the mapping from a role \( r \) onto a set of behaviors,

\[
\text{assigned\_behaviors}(r) = \{ b \in \text{BEHAVS} \mid (r, b) \in \text{AR} \}
\]

On the basis of the above definitions, we define the following function of **avail_session_permissions** to decide whether an access is legal or not. It is the core mechanism of the access decision facility[31] of an access control system.

• **avail_session_permissions** : \( s : \text{ SESSIONS} \) → \( 2^{\text{PERMS}} \), the mapping from a session \( s \) onto a set of permissions,

\[
\text{avail_session_permissions}(s) = \bigcup_{b \in \text{assigned\_behaviors}(r)} \text{assigned\_permissions}(b) \quad (\text{where } r \in \text{session\_roles}(s))
\]

At the moment of a user accesses to a permission, access control system confirms the associations between the user’s session and the permission, and makes an access decision. In core E-RBAC model, the access decision process is consecutive; the system
validates roles of the user, behaviors associated to the role, and permissions related with the behaviors. The function, \textit{avail\_session\_permissions} implicate the consecutive process. The final step of the allowance is that the access control system’s confirmation of the targeted permission is in the set of the validated permissions.

In addition, the access rule is simply derived based on the definition of activated entities. By the definition, a users’ activated roles are derived as follows.

\begin{itemize}
  \item \textit{activated\_roles}(u) = \{ r \in ROLES \mid r \in session\_roles(s) \bullet s = user\_session(u) \}\}
\end{itemize}

The activated roles must be the subset of the authorized roles of the user, and they determine the set of permissions which the user can access. The following activated entities are derived with respect to the activated roles.

\begin{itemize}
  \item \textit{activated\_behaviors}(u) = \{ b \in BEHAVS \mid b \in \bigcup \textit{assigned\_behaviors}(r) \bullet \forall r \in \textit{activated\_roles}(u) \}
  \item \textit{activated\_permissions}(u) = \{ p \in PERMS \mid p \in \bigcup \textit{assigned\_permissions}(b) \bullet \forall b \in \textit{activated\_behaviors}(u) \}
\end{itemize}

Naturally, the following rules are hold between the activated entities and authorized entities.

\textbf{Rule 3.4.1 (Activated Roles, Behaviors, Permissions) :}

\textit{For all user } u \in \textit{USERS,}

\begin{itemize}
  \item A user’s activated roles are the subset of the authorized roles for the user.
  \begin{equation}
  \textit{activated\_roles}(u) \subseteq \textit{assigned\_roles}(u)
  \end{equation}
\end{itemize}
• A user’s activated behaviors are the subset of the authorized behaviors for the user.

\[
\text{activated\_behavior}(u) \subseteq \text{assigned\_behaviors}(u)
\]

• A user’s activated permissions are the subset of the authorized permissions for the user.

\[
\text{activated\_permissions}(u) \subseteq \text{assigned\_permissions}(u)
\]

Finally, we can describe a permitted access as follows:

**Rule 3.4.2 (Permitted Access):**

For a user \( u \) and a permission \( p \) which the user want to access

\[
\text{access}(u, p) \Rightarrow \exists p \in \text{activated\_behaviors}(u)
\]

### 3.4.2 Constrained E-RBAC Model

The constrained E-RBAC model includes additional constraints of Procedural Restrictions (PR). The objectives of the PR are the more precise access control and the prohibition of dangerous executions. PR accomplish the goals with two additions; Orders to constrain behaviors. Identification properties describe whether the ordered behaviors are harmful or not.

Fig. 3.5 shows the conceptual diagram of constrained E-RBAC. We introduce several entities to establish the PR concept as follows.

• Procedural Unit (PU): The extended behavior with an order and an iteration number. If a PU is defined as (‘copy’, 2, 3), it means that the behavior ‘copy’
Figure 3.5: Extended RBAC entities and relations with procedural restrictions.

should be thirdly executed in the procedure and executed twice.

- Procedural Constraint (PC): A procedure consists of a set of PUs. And a PC contains an additional property, called as an identification property, that describes whether the PC is positive or negative. If it is positive, the procedure is executed step by step in a system. If it is negative, the execution of the procedure is blocked before the completion.

- Procedure History (PH): Behavior execution logs. If an access is performed successfully, a PH is recorded. The log record mainly contains such information as behavior, PU, PC, and sessions.

The following definitions summarize the above descriptions.

**Definition 3.4.2 (Procedural Restrictions):**
• \( PU \subseteq (BEHAVS \times N \times N) \), the collection of \((bs, itnum, onum)\), where each \( bs \) is a behavior, and \( itnum \) and \( onum \) are natural numbers. \( itnum \) implies maximum number of repetition for the behavior \( bs \). \( onum \) indicates the order of execution for \( bs \) in the procedural unit.

• \( \text{behavior\_punits} : (b : BEHAVS) \rightarrow 2^PU \), the mapping from behavior \( b \) onto a set of procedural units which include the behavior as a constituent.

• \( PC \subseteq (2^PU \times \text{idprop}) \), the collection of pairs \((punits, idprop)\), where each \( punits \) is a set of procedural units and \( idprop \) is \( \text{POSITIVE} \) or \( \text{NEGATIVE} \).

• \( \text{behavior\_pconstraints} : (b : BEHAVS) \rightarrow 2^PC \), the mapping from behavior \( b \) onto a set of related \( PC \). The returned \( PC \) includes \( PU \) in which behavior \( b \) participates as a constituent.

• \( \text{pconstraints\_idprop} : (pc : PC) \rightarrow \{\text{POSITIVE}, \text{NEGATIVE}\} \), the mapping of a \( PC \) to its identification property.

• \( \text{order\_of\_behavior} : (pc, b) : (PC, b : BEHAVS) \rightarrow N \), the mapping from a behavior to a natural number of a \( PC \). The natural number indicates relative execution order of the behavior in the \( PC \).

• \( \text{is\_the\_last} : (pc, b) : (PC, b : BEHAVS) \rightarrow \{\text{TRUE, FALSE}\} \), the function returns \( \text{TRUE} \) if the behavior \( b \) is the last in the \( PC \), otherwise returns \( \text{FALSE} \).

• \( PH \subseteq \text{SESSIONS} \times \text{ROLES} \times \text{BEHAVS} \times N \times PC \), the collection of \((s, r, b, n, pc)\), where each \( s \) is a session, \( r \) is a role, \( b \) is a behavior, \( n \) is a natural
number which indicates order in a procedural constraints, and \( pc \) is the procedural constraints.

- \( \text{last-onum	extunderscore behavior}(s, pc, b) : (s : \text{SESSIONS}, pc : \text{PC}, b : \text{BEHAVS}) \rightarrow N \), returns the number that indicates relative order of the last executed behavior for the session \( s \) issued \( pc \).

- \( \text{count-itnum	extunderscore behaviors}(s, pc, b) : (s : \text{SESSIONS}, pc : \text{PC}, b : \text{BEHAVS}) \rightarrow N \), returns the number of repeated times of the behavior execution for the session \( s \) issued \( pc \).

With above definitions for PR, an access control system tests each access following the procedure as Fig. 3.6 shows. The access control processes are differed from the identification property of the behavior, and it is also differed from the existence of procedural restriction on the behavior.

If a user accesses to a permission, access control system gets information of the session and the behavior, and check whether a PC is applied to the behavior or not. If a PC is applied, get the identification property of the PC. If the property is negative, deny executions of at least one behavior before all the behaviors in the PC are executed. In case of the positive, execute a behavior if all of the preliminary behaviors were executed and if it does not exceed the number of permitted repetitions.

In the decision process, it is possible that a conflict occurs owing to the reusability of behavior definitions. Assume that a behavior participates in two PCs, \( p_1 \) and \( p_2 \). If both \( p_1 \) and \( p_2 \) are positive PC, it is no problem to permit the executions. If both are
negative, and the behavior is the last one of at least one PC, then the execution should be denied. If one PC is positive, and the other is negative, the denial or allowance is decided according to the policy of the system.

Figure 3.6: Testing procedure of an access with procedural restrictions.
3.5 Additions to E-RBAC

3.5.1 Role Hierarchy

To reflect authority and responsibility in users’ organization, RBAC introduced hierarchies into the role structure[11]. In the relationships between roles, inheritance relations are defined.

Fig. 3.7 shows a simple example of a role hierarchy. The structure of the hierarchy reflects the structure of job-functions in user organization. A role can be defined inheriting another role, then the latter role and the former role are called as ascendant and descendent of each other, respectively. This relationship constructs a role hierarchy.

In the example, a manager is defined by inheriting an employee, and a chief manager is defined by inheriting the manager. It accords to the characteristics of the real-world organization. We can interpret the hierarchy with respect to the member and group
relations. A user who is a chief manager is also included in the manager group, and users whose position is manager are also basically employees of the company. On the other hand, the hierarchy can also be interpreted in terms of the authorities. In the role hierarchy, a descendent having authorities of its ascendent. For example, a manager will have the authority of employees in the system. With the above semantics, the role hierarchies can be defined in the E-RBAC model as follows.

**Definition 3.5.1 (Role Hierarchies):**

- $RH \subseteq \text{ROLES} \times \text{ROLES}$, a partial order relation between ROLES, called as role hierarchies and noted as $\succeq$. $r_1 \succeq r_2$ means users assigned to the role $r_1$ are also assigned to the role $r_2$.

- $r_1 \succeq r_2 \Rightarrow \text{assigned\_users}(r_1) \subseteq \text{assigned\_users}(r_2)$.

- $\text{assigned\_users}(r : \text{ROLES}) \rightarrow 2^{\text{USERS}}$, the mapping from a role $r$ onto a set of users.

  - $\text{assigned\_users}(r) = \{ u \in \text{USERS} \mid \forall r' \in \text{ROLES} \text{ and } r' \succeq r \bullet (u, r') \in \text{SA} \}$

The role hierarchy is the relationship which is defined in the subject-abstraction. However, the functions of the object-abstraction also should be extended with the changed definitions, as follows.

- $\text{assigned\_behaviors}(r : \text{ROLES}) \rightarrow 2^{\text{BEHAVS}}$, the mapping from a role $r$ onto a set of behaviors.
Figure 3.8: The roles that are not separated.

\[ \text{assigned\_behaviors}(r) = \{ b \in \text{BEHAVS} \mid \forall r' \in \text{ROLES} \text{ and } r \geq r' \bullet (r', b) \in AR \} \]

3.5.2 Test of the Relations

Although the hierarchical structure provides several advantages, it augments the complexity of the relationship between roles. When administrate the complex structure of roles, the administrator can have erroneous relationships such as: Fails in separation and Redundant relationship.

The separation of duty is an important security principle, especially in such an environment as a bank. We presented already the examples of static and dynamic separation of duty in Fig. 2.7. With a complex role hierarchy, security administrator can make a mistake by relating two roles directly or indirectly, and hence can break
the separation of duty. Fig. 3.8 shows the roles which are not separated. In the figure, the Role5 and the Role6 are not separated in the hierarchy. The Role6 and the Role9 are also not separated in terms of if the owner of the Role1 can have both roles.

Redundant relationships are easily generated in relating roles without attention, and bring situation against security of the system. For example, job positions and functions are defined in an organization hierarchy of a bank as:

- Account Management Director: The chief officer of account management.
- Auditor: The inspector of transactions. Auditor analyzes transaction logs of the bank and finds erratic or illegal data. The superior of the auditor is the account management director.
- Account Manager: The manager of account service. Account manager takes in charge of the managing account information and customer information. The superior of the account manager is the account management director.
- Transaction Agent: The agent records, classifies, and maintains transaction data. The agent supports the tasks of account manager and auditor.

By the current definition of organization hierarchy, the auditor and the account manager are directed by the account management director. Also, the transaction agents are directed by the auditor and the account manager.

However, the reorganization of the hierarchy is performed to separate auditing from the account management. Referring the defined relationships, the security officer of the system separate the auditor from the account management director. As the graphical
Figure 3.9: A redundant relationship.

notation of the hierarchy in Fig. 3.9, the security officer removes the link 'x' between the role of account management director and its descendent, the auditor. However, against the security officer’s intention, the account management director still has a relationship with the auditor after the modification. This redundant factor is not easily found in a complex organizational structure.

By the way, if we broaden our visions from the role entities to the other entities, all entities from users to permissions are related each other in E-RBAC. The access rules are finally the relations between users and permissions. We can figure out the whole authorized relations of a system as Fig. 3.10. There are links from users to permissions in the directed graph. The links pass through the intermediate entities, and they represent the authorizations.

The previous two erroneous relationships are also found in the graph.
• Fails in separation: If the permissions, $b$ and $c$ are prohibited from standing together, current graph does not support the separation. The permissions can be authorized to a user.

• Redundant relationship: As an example, there is a redundant path between the node $a$ and the node $b$. Therefore, it is possible that the security officer fails to remove the permission of $b$ from a user.

The errors can make security problems within all entities. To reduce these erroneous relations, we can establish testing rules. At first, we define a set that has all entities and relations of the E-RBAC model.

**Definition 3.5.2 (Entity-Relation Set, ER):**
A finite set $ER$ is defined as follows.

- $U$, $R$, $B$, $P$: users, roles, behaviors, permissions, respectively.

- $SA \subseteq U \times R$, $OA \subseteq B \times P$, $AR \subseteq R \times B$.

- $RH \subseteq R \times R$, $BH \subseteq B \times B$.

- $ER = \{ x \mid x \in U \text{ or } x \in R \text{ or } x \in B \times P \} = U \cup R \cup B \cup P$.

- $u1 \in U$, $r1, r2 \in R$, $b1, b2 \in B$, $p1 \in P$.

- A binary relation $\preceq$ is defined as
  - $p1 \preceq b1$, if there is a relation between the behavior $b1$ and the permission $p1$ in $OA$.
  - $b1 \preceq b2$, if there is a relation between the behavior $b1$ and the behavior $b2$ in $BH$, and $b2$ includes $b1$.
  - $b1 \preceq r1$, if there is a relation between the behavior $b1$ and the role $r1$ in $AR$.
  - $r1 \preceq r2$, if there is a relation between the role $r1$ and the role $r2$ in $RH$, and $r2$ inherits $r1$.
  - $r1 \preceq u1$, if there is a relation between the role $r1$ and the role $u1$ in $RH$.

Then, we can constitute a graph from $ER$, and it is same to the graph of Fig. 3.10, but the direction of the arrows are inverted. The graph called as $ER$-graph is shown in Fig. 3.11.
Let’s assume that two properties are satisfied in the set ER. First, there is no cycle in ER-graph. It means that there is no security officer’s mistake of making a cyclic relationship within role-hierarchy or behavior relations. Second, there is no dangling node in ER-graph. That is, there are no roles, behaviors, permissions that are defined but not assigned. The example diagram of the cycle and dangling node of the ER-graph are shown in Fig. 3.11.

In addition, we introduce the two special entities of $\top$ and $\bot$ into our ER set. For convenience, we regard the element $\top$ represents the group of all users. Therefore, $\top$ inherits all users. On the other hand, $\bot$ can be regarded as no permission. All
permissions inherits \( \perp \).

- \( \top \) (Top): \( \forall u \in U, u \leq \top \)

- \( \bot \) (Bottom): \( \forall p \in P, \bot \leq p \)

For the elements \( \forall x, y, z \in ER \), the binary relation \( \preceq \) satisfies following three properties.

- \( x \preceq x \): The relation is reflexive.

- if \( x \preceq y \) and \( y \preceq z \), then \( x \preceq z \): The relation is transitive.

- if \( x \preceq y \) and \( y \preceq x \), then \( x = y \): The relation is anti-symmetric.

Therefore, \( (ER, \preceq) \) is a partially ordered set, or poset.

In the poset \( ER \), we can define a greatest lower bound and a least upper bound elements[32, 33] as follows.

**Definition 3.5.3 (Greatest lower bound, Least upper bound)**:

Let \( (ER, \preceq) \) be a poset with \( A \subseteq ER \). An element \( x \in ER \) is called a **lower bound** of \( A \) if \( x \preceq a \) for all \( a \in A \). Likewise, an element \( y \in ER \) is called an **upper bound** of \( A \) if \( a \preceq y \) for all \( a \in A \).

An element \( x' \in ER \) is called a **greatest lower bound** (glb) of \( A \) if it is a lower bound of \( A \) and if for all other lower bounds \( x'' \) of \( A \) we have \( x'' \preceq x' \). Similarly \( y' \in ER \) is a **least upper bound** (lub) of \( A \) if it is an upper bound of \( A \) and if \( y' \preceq y'' \) for all other upper bounds \( y'' \) of \( A \).
In addition, we also define operations for the greatest lower bound and the least upper bound as follows.

**Definition 3.5.4 (Operation $\lor$, $\land$):**

We define binary operations **join** (noted as $\lor$) and **meet** (noted as $\land$) on the non-empty set ER by

- (for all $x, y$ in ER) $x \lor y$: the least upper bound of $\{x, y\}$, and

- $x \land y$: the greatest lower bound of $\{x, y\}$

Similarly we write $\lor S$ (the **join of** S) and $\land S$ (the **meet of** S) for the least upper bound of the set S and the greatest lower bound of the set S, respectively.

With the defined operation, we can find the erroneous relationship as follows.

- **Fails in separation:** For $x, y \in \text{ER}$, if the separation of $x$ and $y$ is defined, then ‘$x \lor y = \top$’ must be hold. Otherwise, it fails in separation of the elements. Similarly, for the set of separated entities $S$, $\lor S$ must be $\top$.

For example, in Fig. 3.11, $b \lor c$ is not $\top$. Therefore, the permissions b and c is not separated for a role.

- **Redundant relationship:** For $a, b, c \in \text{ER}$, where $b \preceq a$ holds, to test the redundant relationship between $a$ and $b$, the set of the direct parents of $b$, $P$ should be calculated (that is, for all $p \in P$, $b \preceq p$ is defined). For If $\lor P$ is $c$, and $c \preceq a$, then there are two or more directed paths between $a$ and $b$.

For example, in Fig. 3.11, we can get $\lor P$ is $a$. Because $a \preceq a$ holds, there are two or more paths between $a$ and $b$. 

Figure 3.12: Example definitions of positive/negative procedure.

3.6 Analysis

As mentioned in Sect. 2.4, the more accurate access control is possible with the procedural restrictions. Moreover, the extended model can refuse attacks that consist of ordinary operations. It has been unable to limit those attack trials in traditional access control of RBAC.

Fig. 3.12 shows example definitions of behavior with positive and negative PC. In the left figure, the procedures of a log file management are defined as positive PCs. For the log file manipulation, the permissible executions are ‘open - close’, ‘open - append a log - close’, and ‘open - read log data - close’. But, it is prohibited to write some data to the logfile after a user locates the file pointer to an intended position by reading several records from the logfile.

In the right figure, a set of behavior is defined as a negative PC. Some version
of sendmail program has a security flaw leaking administrator’s authority. The race condition attack\cite{25} can be used for the intrusion, and the attack tries to do ‘unlink’ and ‘symlink’ a temp file to ‘.rhosts’ repeatedly. Therefore, if it is prohibited that the unlink/link repetition by defining negative PC, the system is free from race condition attacks.

In the extended access control, the main decision process is similar to that of previous RBAC. In RBAC, if a user accesses an object, access control system check whether the permission on the object is associated with the user’s acquired role. In the extended method, access control system checks if the behavior has the permission on the object is associated or not with the user’s acquired role.

However, the extended access control is different with previous RBAC. In the previous concept, roles are the neutral entities which are placed between users and permissions. It can be interpreted as both, and it is the figuration of the mapping between users and permissions. On the contrary, in the extended concept, roles and behaviors are the abstraction for users and permissions, respectively. They are interpreted as subjects and objects separately. And, in extended RBAC, there are no entities between roles and behaviors, and still associations are exist instead. It is different from the RBAC’s visualization of the mapping. In other words, it is possible that the roles can be considered as groups with hierarchy, and behaviors as functions with nested structures.
3.7 Summary

In this chapter, we described an extended RBAC concept and model. By the extension, access control not only bases instant access control information but also continuous access information. Thus, access control efficiently limits attack trials which consist of allowed operation. The race condition attack was presented as an example of those attacks trials. The race condition attack is a kind of attack using time-of-check-to-time-of-use (TOCTTOU)[23] vulnerabilities. The attacks exploit inconsistencies of synchronization in multi-user environments and they use ordinary operations. We expect the extended method can deny more attacks based on similar mechanism.

Extended RBAC useful to prevent various known-attacks because the description of attack sequences is supported in procedural restrictions. It means the function of access control is expanded to the coverage of IDSs, though its ability has limitations due to performance overhead. However, for some attacks, access control provides more reliable solutions, because access control is performed at the kernel-level not being by-passed. It is different from IDSs which are executed in the application level.
Chapter 4

Coloured Petri Net Formal Model for E-RBAC

4.1 Introduction

Formal methods have been applied to the development process of security solutions. Especially, formal specification and verification methods are the requirements for higher assurance.

Trusted Computer System Evaluation Criteria[7] (TCSEC, or called as “Orange Book”) is a computer security standard issued by National Computer Security Council of US. The guidelines of TCSEC are often used to evaluate the security level of trusted operating systems. TCSEC graded the security levels of security products into six classes and stated the requirements for the classes. Following table shows the levels and summarizes their characteristics.

Table 4.1: TCSEC Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>The lowest level. Not secure.</td>
</tr>
<tr>
<td>C1</td>
<td>User log-on is required, but allows group ID.</td>
</tr>
</tbody>
</table>
Table 4.1: TCSEC Classes (Continued)

<table>
<thead>
<tr>
<th>Class</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td>Individual log-on is required. The log-on process is authenticated by password and it is logged. (General Unix and Linux operating systems.)</td>
</tr>
<tr>
<td>B1</td>
<td>Mandatory Access Control is enforced. Clearance levels are required.</td>
</tr>
<tr>
<td>B2</td>
<td>Trusted Paths are guaranteed. Assurances are provided. Clearance cannot be downgraded</td>
</tr>
<tr>
<td>B3</td>
<td>Characterized by viable mathematical model</td>
</tr>
<tr>
<td>A1</td>
<td>Proved by mathematical model</td>
</tr>
</tbody>
</table>

In Fig. 4.1, the more detail requirements are listed[46]. As the figure shows, formal model of the security policy and the verification of system design are required. The system design should be consistent with the model, and it should be proven by formal methods.

Formal methods can help the development of trusted operating systems in following different ways. First, formal specifications help to understand several parts of trusted operating systems. The requirements, security policy, access control policies can be formally specified with mathematical notations. The formal specification reduces am-
Figure 4.1: A summarized chart of the Trusted Computer System Evaluation Criteria

biguities and makes the ideas be delivered clearly. Particularly, formal methods are
useful to integrate multiple access control schemes in trusted operating systems. So
far, several implementations of security kernels have newly introduced mandatory poli-
cies into Unix or Linux kernels where discretionary policies are enforced. Some of the
systems have used formal methods to specify the unified and conflict-free enforcements
of the policies[19].

Second, the data structures and functions in trusted operating system implemen-
tations can be formally specified, and we can verify how well the specification meets the requirements. Besides accomplishing correct implementation, we can develop and reliable software and can improve the productivity. In the field of software engineering, they have used formal methods for the software productivity.

Third, the formal method can be used in security administration. We can specify the configuration and evaluate security-related properties of a trusted operating system mathematically.

As previously mentioned in Sect. 2.2, access controls are realized by the following steps; describe a policy, specify the policy with model, implement a framework to enforce the policy, and administrate the implemented system. It is similar to defending a fortress. To defend the fortress, we contrive a plan to guard, build traps and barriers around the castle, and assign soldiers to appropriate positions. Security administration is the process corresponding to the assignments of soldiers.

The security administration is a work of making a security rules and managing the system configuration. It is regarded as important as the system design and implementation. The security administration is also an independent process from the design and implementation. The administration decides the security of the system after an implementation is installed. Even though the implementation is perfect, poor administration makes the implemented framework be useless.

The most important work of the administration is building and managing security configuration correctly. If there is an error or a fault in security configuration, it can derive security flaws of the whole system. For example, if the security administrator of
the system makes a mistake during access rule establishments, it leads following bad results.

- unauthorized access: if there is a connection between unauthorized user (such as guest) and an important permission (such as write operation to the /etc/password file) in the access rule, then the system will be exposed to the unauthorized accesses. The single broken security rule can bring the crumbled security of the whole system.

- denial of service (DOS): if no access rule is defined between a legitimated user and an expected operation, the situation leads to the denial of service.

Therefore, finding and correcting faults in a security configuration is an important process of the security administration. However, it is too hard to find faults in the configuration using manual or trial-and-error approach.

An operating system is huge system software. There are many entities are exist and they are concurrently run interacting each other. The configurations of the system have many terms and the values of the terms will affect each other during the interactions. Therefore, it is hard to evaluate the configuration manually. Instead of the manual approach, formal methods can be used. Formal methods guarantee a formally verified fact always holds for all possible cases. Moreover, some formal methods support automatic computerized tools for an evaluation.
4.2 Related Work

Various formal models have been used to describe access control schemes. In case of discretionary access control, it has not any specific formal model. It is hard to prove or validate the discretionary access control mechanisms with formal methods exactly. The discretionary policy has not well-defined access control principles.

On the contrary, mandatory access control and RBAC schemes have been described with well-defined formal methods. BLP[12] model was defined with finite state machine approach. RBAC have been mainly described with set notations.

The access control models are described with the other formal specification methods such as Z[55] and Alloy[56]. The Z and Alloy specifies a concept or a system based on proof-based logic. Z is a formal specification language which is describes computer-based systems. Z is based on set theory and first order predicate logic. It is now defined by an ISO standard. Z has been used to specify access control models and implementations. By specifying models, it explains the idea and properties of the access control concepts clearly. By specifying implementations, it shows the functions and states of the system without ambiguity. The security kernel of DTOS project specified by Z language[14]. Alloy is the subset of Z language. It has user-friendly syntaxes consist of ASCII characters. We introduce an example of the Alloy specification later in Sect. 4.2.2.

On the other hand, several state machine based specification models such as finite state machine and Petri Nets have been used to describe access control models and systems. The state machine based (or state-transition based) approaches provide some
advantages to which we pay attention in this paper. It is able to describe a system graphically with the state machine based models. Especially, the models efficiently express states of a system and their changes. In addition, they usually provide various automatic verification tools. Therefore, we will apply the state machine based formal model to describe our extended access control system. In our extended access control, the execution sequences are important and they are specified appropriately with the state machine based approach. Moreover, to use a formal approach in security administration, the automatic verification tools are very useful. As an example of the state-machine based modeling, we introduce the Petri Net based specification in Sect. 4.2.3.

4.2.1 Access Control Models

BLP Model

Bell and LaPadula (BLP) model have been used in governments and military systems because BLP enforces very strong security policy and controls information flows. BLP model is based on state-transition model. It consists of following elements.

<table>
<thead>
<tr>
<th>Set</th>
<th>Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>$S_1, ..., S_n$</td>
<td>the set of subjects</td>
</tr>
<tr>
<td>O</td>
<td>$O_1, ..., O_m$</td>
<td>the set of objects</td>
</tr>
</tbody>
</table>
Table 4.2: The Elements of Bell-Lapadula Model (Continued)

<table>
<thead>
<tr>
<th>Set</th>
<th>Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>$C_1, ..., C_p$ and $C_1 &gt; C_2 &gt; ... &gt; C_p$</td>
<td>the classifications which are the clearance levels of the subjects and classification levels of the objects</td>
</tr>
<tr>
<td>K</td>
<td>$K_1, ..., K_r$</td>
<td>need-to-know categories or compartments of an organization</td>
</tr>
<tr>
<td>A</td>
<td>read, write, copy, append, owner, control</td>
<td>the set of access modes</td>
</tr>
<tr>
<td>R</td>
<td>$R_1, ..., R_u$</td>
<td>the access requests from subjects to objects</td>
</tr>
<tr>
<td>D</td>
<td>yes, no, error</td>
<td>access decisions</td>
</tr>
<tr>
<td>T</td>
<td>1, 2, ..., $t$, ...</td>
<td>indices which identifies discrete time</td>
</tr>
<tr>
<td>P()</td>
<td></td>
<td>power set</td>
</tr>
<tr>
<td>$\alpha^\beta$</td>
<td></td>
<td>all functions from $\beta$ to $\alpha$</td>
</tr>
<tr>
<td>F</td>
<td>$C_s \times C_o \times (PK)^s \times (PK)^o$</td>
<td>classification and need-to-know noted as $f = (f_1, f_2, f_3, f_4)$</td>
</tr>
</tbody>
</table>
Table 4.2: The Elements of Bell-Lapadula Model (Continued)

<table>
<thead>
<tr>
<th>Set</th>
<th>Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>$M_1, \ldots, M_s$</td>
<td>access matrices and the $(i,j)$ entry of a matrix $M_k$ shows $S_i$’s permitted action on $O_j$</td>
</tr>
<tr>
<td>V</td>
<td>$P(S \times O) \times M \times F$</td>
<td>states</td>
</tr>
<tr>
<td>Z</td>
<td>$V^T$</td>
<td>state sequences</td>
</tr>
</tbody>
</table>

With the elements, a state of a system is defined as follows.

**Definition 4.2.1 (State of a system):** A state of a system $v \in V$ is defined as a 3-tuple $(b, M, f)$, where

- $b \in P(S \times O)$; which subjects access to which objects at the state of $v$.
- $M \in M$; the entries of the access matrix at the state of $v$.
- $f \in F$; the clearance level of all subjects, the classification level of all objects, the needs-to-know associated with all subjects, and the need-to-know category of all objects at the state of $v$.

After this definition, operations in the system are the relation of states and transitions as $W \subseteq R \times D \times V \times V$. Then, we can define the system as follows.
Definition 4.2.2 (System) : The system $\Sigma (R,D,W,z_0) \subseteq X \times Y \times Z$ is defined by

$(x, y, z) \in \Sigma (R,D,W,z_0)$ if and only if $(x_t, y_t, z_t, z_{t-1}) \in W$ for each $t \in T$, where $z_0$
is a initial state of $(\emptyset, M, f)$

- $(x, y, z) \in \Sigma (R,D,W,z_0)$ is the appearance of the system
- $(x_t, y_t, z_t, z_{t-1}) \in W$ is the action of the system

Therefore, we can define a compromise state and the security condition as follows.

Definition 4.2.3 (Compromise State) : $v = (b, M, f) \in V$ is a compromise state if there is an ordered pair $(S, O) \in b$ such that

- $f_1(S) < f_2(O)$, or
- $\neg f_3(S) \supseteq f_4(O)$.

Definition 4.2.4 (Security condition relative to $f$ (SC rel $f$)) : $(S, O) \in S \times O$ satisfies the security condition relative to $f$ (SC rel $f$) if

- $f_1(S) \geq f_2(O)$ or
- $f_3(S) \supseteq f_4(O)$.

Finally, we can define a secure state, a secure state sequence, a secure appearance, and secure system step by step as follows.

- A state $v = (b, M, f) \in V$ is a secure state if each $(S, O) \in b$ satisfies SC rel $f$.
- A state sequence $z \in Z$ is a secure state sequence if $z_t$ is a secure state for each $t \in T$.
Figure 4.2: The conceptual diagram of RBAC concept.

- An appearance \( (x, y, z) \in \sum(R,D,W,z_0) \) is a secure appearance if \( z \) is a secure state sequence.

- A system \( \sum(R,D,W,z_0) \) is a secure system if every appearance of \( \sum(R,D,W,z_0) \) is secure.

**RBAC Model**

RBAC models have focused on the description of components and the relations between the components. This is different from the mandatory access control models. In the mandatory schemes, all accesses most follow the few but strong principles like as the three security properties of BLP model which is mentioned in Sect. 2.2.2. However, the access rules of RBAC are defined by the form of relations between access subjects and access objects, that is users and permissions in RBAC. There are the intermediary entities; roles are intervening in the relationship. The security officer of the system establishes the relations reflecting context information of the system,
and an access is granted if there is a relation between the subject and the object. In addition, access rules are dynamically changed, on the contrary to the immutable principles of the mandatory schemes. As a result, the security rules of RBAC are extremely complex comparing to the mandatory schemes. The RBAC model deals with the components, relationships, necessary functions which traces the relationships, and various constraints which are given to the accesses. The conceptual diagram of RBAC is shown in the Fig. 4.2.

The specification of the RBAC model starts from the core RBAC model. It is more specified with additional features in hierarchical RBAC and constrained RBAC models.

The core RBAC is defined in the NIST’s standard[11] as follows.

**Definition 4.2.5 (Core RBAC) :**

- **USERS, ROLES, OPS, and OBS:** the set of users, roles, operations, and objects.
- **UA ⊆ USERS × ROLES:** user-to-role assignment relation.
- **assigned_users(r) = u ∈ USERS | (u, r) ∈ UA**
- **PRMS = 2^{OPS×OBS}:** the set of permissions.
- **PA ⊆ PRMS × ROLES:** permission-to-role assignment relation.
- **assigned_permissions(r) = p ∈ PRMS | (p, r) ∈ PA**
- **Op(p : PRMS) → op ⊆ OPS:** permission-to-operation mapping.
- **Ob(p : PRMS) → ob ⊆ OBS:** permission-to-object mapping.
• **SESSIONS**: the set of sessions.

• **user_sessions**(*u : USERS*) → 2^{SESSIONS}

• **session_roles**(*s_i*) ⊆ *r ∈ ROLES | (session_users(*s_i*, *r*) ∈ UA

• **avail_session_perms**(*s*): the available permissions for a user in a session

\[= \bigcup_{r \in session\_roles(s)} \text{assigned\_permissions}(r)\]

Hierarchical RBAC introduces the concept of hierarchy in the role-to-role relationship. A role can inherit the other role, then the former role succeeds the permissions of the latter role. By the permission inheritance, the concepts of authorized users and authorized permissions are extended. They are formally specified as follows.

• **RH** ⊆ **ROLES** × **ROLES**: a partial order relationship on Roles.

  if *r_1* ≥ *r_2*, then \( \text{authorized\_permissions}(r_2) \subseteq \text{authorized\_permissions}(r_1) \) and \( \text{authorized\_users}(r_1) \subseteq \text{authorized\_users}(r_2) \)

• **authorized_users**(*r : ROLES*) = *u ∈ USERS | r' ≥ rand(u, r') ∈ UA

• **authorized_perms**(*r : PRMS*) = *p ∈ PRMS | r' ≥ rand(p, r') ∈ PA

Constrained RBAC adds the separation of duty relationships into the roles. Separation of duty relationship prevents a user from having too much permissions as mentioned in Sect. 2.2.3. There are two kinds of separation of duty relationships of static separation of duty and dynamic separation of duty. They are specified as follows.
• SSD ⊆ 2^(ROLES × N) is collection of pairs (rs, n) in Static Separation of Duty, where each rs is a role set and n is a natural number. No user is assigned to n or more roles from the set rs in each (rs, n) ∈ SSD. This property is specified formally as, ∀(rs, n) ∈ SSD, ∀t ⊆ rs : | t | ≥ n ⇒ ∩_{r ∈ t} = assinged_users(r) = ∅ (, where t is a subset of roles in rs)

• DSD describes the property of no simultaneous activation of roles. No subject can activate n or more roles from rs in each dsd ∈ DSD.

∀ rs ∈ 2^ROLES, n ∈ N, (rs, n) ∈ DSD ⇒ n ≥ 2 ∧ | rs | ≥ n, and

∀ s ∈ SESSIONS, ∀ rs ∈ 2^ROLES, ∀ role_subset ∈ 2^ROLES,

∀ n ∈ N, (rs, n) ∈ DSD, role_subset ⊆ rs,

role_subset ⊆ session_roles(s) ⇒ | role_subset | < n

Based on the basic definition of RBAC model so far, the functional specification can be carried out. The functional specification is the implementation guideline of RBAC systems. In the development phase, the specification shows a clear blue print of the implementation, and hence reduces misunderstanding in the communication between developers. Moreover, it is able to test logically whether the system has a logical fault in its operation before the system is implemented. However, they are specified based on the context of a target system, hence the specification can be different in systems. In the NIST’s standard for RBAC, the specified functions are categorized as follows.

• administrative functions: The functions create and manage the sets and relations of RBAC components.
• supporting system functions: The functions related to the runtime features of RBAC system such as session management and access decisions.

• review functions: The functions inquire the current states of the sets and relations.

For more formalized specification and the test of the correctness of the system, RBAC have been specified in particular formal methods. Especially, the formal specification language Z has been used to specify several access control policy and implementations such as the NIST’s standard RBAC, the DTOS project[14] and so on. The following example schema is the specification of the AddUser, one of the administrative functions of the NIST’s standard RBAC.

\[ AddUser(user : NAME) \]

\[ user \notin USERS \]

\[ USERS' = USERS \cup \{ user \} \]

\[ user_sessions' = user_sessions \cup \{ user \mapsto \emptyset \} \]

The specification explains that the function of the AddUser; if a name is given as the input of the function, it adds the name to the set of users, and add a new null session for the user.

4.2.2 Alloy specification of the core E-RBAC model

Alloy is a light-weight modeling and analysis language developed by MIT[56]. Alloy is a simple and compact. The notations of the language are ASCII characters, and hence the syntax is rather easy to learn and use than other languages that have complicated
math symbols. Although it has simple syntax, Alloy is powerful and flexible to describe complex applications. It is able to modeling a huge system consisting thousands lines of codes with few lines of Alloy. Moreover, Alloy provides an analyzing tool, Alloy Analyzer. The tool simulates model and tests properties of the model by generating behavioral states from input statements and showing counter examples.

The following example Alloy codes describing the basic elements of the core E-RBAC. The syntax of Alloy is very intuitive, and hence it is easily understandable that following codes describe the relations between entities.

domain{ Users, Roles, Sessions, Behaviors, Permissions, NBehaviors }

state{

ur_assignment: Users -> Roles

rb_assignment: Roles -> Behaviors

bp_assignment: Behaviors -> Permissions

us_assignment: Users! -> Sessions

sr_assignment: Sessions -> Roles+

rr_hierarchy: Roles -> Roles

bb_hierarchy: Behaviors -> Behaviors

nb_set: NBehaviors -> Permissions

}

Additionally, several invariants can be declared to describe the property of the E-RBAC model. For example, we can give partial order properties to role-to-role relationship and behavior-to-behavior relationship.
inv partial_order{
    all r:Roles | r in r.rr_hierarchy // reflexive
    all r | r.*rr_hierarchy in r.rr_hierarchy // transitive
    all r1: Roles, r2: Roles | (r1 in r2.rr_hierarchy && r2 in r1.rr_hierarchy)
    -> r1 = r2 // anti-symm
}

inv authorization{
    all u:Users | all r:Roles | r in u.us_assignment.sr_assignment
    -> r in u.ur_assignment
}

The authorized user-to-role relationship can be declared as an invariant of the system. If a current authorization for a role is given to a user who have activated the role in a session.

inv negative_behavior{
    all b:Behaviors | b in b.bb_hierarchy // reflexive
    all b | b.*bb_hierarchy in b.bb_hierarchy // transitive
    all b1: Behaviors,b2: Behaviors |
    (b1 in b2.bb_hierarchy && b2 in b1.bb_hierarchy) -> b1 = b2 // anti-symm
}

We should distinguish the harmful behaviors and permitted behaviors. The difference is described with the following invariants.
all u:Users | all nb: NBehaviors | some(nb.nb_set -
   u.ur_assignment.rb_assignment.bp_assignment)
}

By defining other invariants properties to describe the characteristics of E-RBAC, we can specify the extended access control model. After the specification, we can test several properties with the automatic analysis tool, Alloy Analyzer.

Fig. 4.3 shows a part of the result of modeling core E-RBAC model. In the figure, the diamonds are negative behaviors and the rectangles are positive behaviors. No positive behaviors are equivalent to negative behaviors. With the result, we can confirm the prohibition of roles from consisting negative behaviors.

However, Alloy is a declarative model, and hence it is not appropriate for our objective. The execution sequences of our extended model can be regarded as programs. They will be modeled efficiently with instructive languages or state-transition based model. The concept of the order is not expressed in the specification. This analysis is not appropriate for the constrained E-RBAC model. From the next section, we will
introduce state-transition based models for the constrained E-RBAC model.

4.2.3 Petri Net Models

The previously introduced E-RBAC models in Sect. 3.4 and Alloy modeling results are described with the set based notations. The specifications are enough to understand the concept of E-RBAC. However, it is not a method to describe the sequence of attack operations conveniently. The attack sequences have state and changes of states, therefore it is not easy to specify them with the set notations. For the specification of constrained E-RBAC, more abstract data structures and primitive operations should be defined to express ordered executions. However, it is not easy and it makes the specification be more complex. It is hard to identify procedural components hard to describe the functions. Moreover, modelers should be familiar with the mathematical notations for the specification. Besides, the set-notation approach is difficult to be verified automatically.

There is the other type of formal approach: the state machine based approach. It is very intuitive to model the execution sequences of the constrained E-RBAC with the state-transition based modeling methods. Graphical representations can be used for easy specification and analysis. Moreover, it can be verified with automatic model checkers or verifiers. Therefore, we pay attention the state machine based formalisms.

In this section we introduce two access control models. They are based on Petri Nets and Coloured Petri Nets. Petri Nets[62] are famous state machine based formal models. Petri Nets are appropriate to model concurrent systems, and hence they
have been introduced into the various research areas of concurrent and asynchronous systems such as agent systems, workflow systems, hardware chip design, real-time operating systems, and so on. Moreover, Coloured Petri Net (CPN) has been used to analysis safety-critical systems[53, 54] where we can learn similar analysis properties and verification techniques.

A Petri net is a directed graph with two types of nodes: places and transitions. A directed arc exists between two different types of nodes. Another important element of the Petri net is the Token. Tokens are the markers that are distributed in places of the net. The state of the system is expressed by the distribution of tokens. Places and transitions are depicted as a circle and bars. Tokens are represented by a dark dot on the places. For more detail of Petri Net formalism can be referred in other texts[59, 60]

The base of all Petri net models is the definition of a net[61].

**Definition 4.2.6 (A Net)**: A net is a triple $N = (P, T, F)$ where

- $P$ is a set of places,
- $T$ is a set of transitions, disjoint from $P$, and
- $F$ is a flow relation $F \subseteq (P \times T) \cup (T \times P)$ for the set of arcs.

Petri Nets have been already applied for information flow analysis. Varadharajan[57] proposed an extended Petri Net model for information flow analysis, and Juszczyszyn[58] suggested a secure Coloured Petri Net (SCPN) by extending Coloured Petri Net. Colored Petri Net is a kind of Petri Net supporting type manipulations.

The extended Petri Net is defined as 13-tuple of
\[ N = (P, T, SC, Tokens, Data, OP, CLSFN, I, O, psec, transop, transcls, M_0) \]

- **P**: Places represent information channels.
- **T**: Transitions correspond to the processes in a system.
- **SC**: The set of security classes which have partial order between the classes.
- **Tokens**: The set of tokens corresponds to the information present on the channels.
- **OP**: The set of operations.
- **CLSFN**: The functions represent how the processes affect the security class of tokens.
- **I**: Input function. A mapping from a transition to a bag of places.
- **O**: Output function. A mapping from a transition to a bag of places.
- **psec**: The function associates a set of security classes with a place.
- **transop**: The function associates an operation with each transition.
- **transcls**: The function associates with each transition with the security class of the output tokens.
- **\( M_0 \)**: A bag of initial marking.

The extended Petri Net is defined for the control of information flows. Tokens and places are labeled with security classes. When a token passes a transition, the labels
of the token and input places are compared, and the label of token is evaluated. The label of token is also compared to the output place of the transition when tokens out.

In the extended Petri Net the secure states and secure chaining of the states are defined canonically. A system which enforces mandatory access control and have security classes can be specified with the extended Petri Net, and the security properties of the system can be tested.

SCPN is another version of the extended Petri Net. However, SCPN bases Coloured Petri Net formalism instead of the Petri Nets. Coloured Petri Net is a high level Petri Net introducing type manipulation so that describing a system and properties with compactness (See Fig. 4.4).

SCPN is defined as 9-tuple of

$$N = \langle CS, P, T, Arc, Node, Colour, Guard, E, Init \rangle$$

- $CS = \{\text{sclass, dat, stoken, scplace, sctrans, Token}\}$.

- sclass is a set of security classes.
- dat is possible data items.

- sctoken and scplace are the security class of token and place respectively.

- sctrans is the value of "secure" or "insecure"

- Token is a tuple ⟨sctoken, dat, scplace, sctrans⟩.

- P, T, and Arc are the set of places, tokens and arcs, respectively.

- Node is a function from arcs to the sets of a places and a transition.

- Color is a function from places to CS.

- E is the arc expression function. The function maps each arc into an expression which manipulates colors (defined types).

- Init is the initialization function that maps each place to an expression.

A SCPN net has labeled information and channels. Channels evaluate the security label of information when they process the information. SCPN defines secure states and transitions similar to the extended Petri Net.

We reviewed two Petri Net access control models in this section. However, those formal models are not reasonable for the E-RBAC modeling also. The information flow models interpret tokens, places, and transitions of Petri Nets as objects, channels, subjects respectively, and the firing sequence models the flow of information. To model Procedural Constraints of the E-RBAC, we should arrange objects and define their relationships. But, it is impossible since objects are represented by tokens. Moreover, the models have no component to describe traditional access control information.
The information flow models are established on fundamentally different concepts and objectives from the access control models.

4.3 Constrained Coloured Petri Net

In this section, we propose an interpretation of Coloured Petri Net (CPN)\[64, 65\] with security considerations for the formal specification of E-RBAC.

CPN is an efficient method to model the multi-user system in which the E-RBAC policy is enforced. Therefore, we adopt the CPN formalism for the E-RBAC specification. Additionally, CPN provides following important advantages\[64, 65\]. The advantages are reasons why we choose CPN formalism among the various state machine approaches.

- CPN has an explicit description of both states and actions: To model E-RBAC policy, we should express two main security conditions. First, the access conditions defined by the access relation information. It should be specified and tested in the formal model that the relations between E-RBAC entities as users, roles, behaviors, and permissions. The relations and testing functions have been described with set-based notations. In CPN, type manipulation expressions are supported, and we can describe the relations and functions with the types and type expressions. Second, the execution sequences of operations. The sequence of operations are easily expressed with the state and transitions of CPN. Therefore, CPN is the very appropriate method to model E-RBAC by providing two kinds of formal methods - Proof based method and State machine based method.
CPN support hierarchical structures: In E-RBAC several permitted execution sequences and dangerous execution sequences are described. They are dynamically modified from the context information. New permitted procedures can be defined in a model. A new known-intrusion procedure can be added to the model. Therefore, the supporting of hierarchical structure provides an advantage for the dynamic security policy enforcement.

CPN has computer tools supporting their drawing, simulation, and formal analysis. One of the applicable areas of our formal model is the security administration. To reduce administrative burden in the administration, and hence eliminate faults and errors in security configurations, computerized automatic tools are very useful.

4.3.1 Basic Concepts

We extended CPN components and translate each of them into the elements of the E-RBAC. In the proposed Constrained Coloured Petri Net (CCPN), tokens, places, and transitions are interpreted as subjects, objects, and Access Enforcement Functions (AEF)[31], respectively.

A subject is presented as a token which itinerates a net as the subject accesses objects. An access of the subject to an object is a movement of a token from one place to the next place. To move, the token must be enabled and fired at the transition which is between the places. When the token is being fired, the access conditions are checked at the pre-place, the transition, and the post-place based on the traditional
access matrix information. That is, the subject should have authorized roles, and the access should not be against the principle of SODs. Fig. 4.5 shows the basic structure of the CCPN model.

On the other side, Procedural Constraints (PC) of the extended E-RBAC is represented as a subnet. An operation execution sequence is represented as the arranged places. The token’s itinerary is along the places under PC. A negative PC leads unauthorized accesses to an isolated state, and prohibits further accesses. Or, a positive PC forces accesses to be occurred in predefined order. As a result, CCPN expresses the sequence of accesses relying on a state-transition structure, as well as access conditions of access matrix model.

4.3.2 Constrained Coloured Petri Net

Definition 4.3.1 (Constrained Coloured Petri Net, CCPN) :

A Constrained Coloured Petri Net is defined as 10-tuple,
CCPN = < Σ, P, T, A, N, C, G, E, I, X >

- Color set Σ = {Uid, Sid, Oid, Mode, Op, Roles, R_tok, R_p, Auth, Token, SSOD, DSOD}
  - color Uid, Sid, Oid = integer; Identifier of users, sessions, objects, respectively.
  - color Mode = { m₁, ..., mₙ }; Access modes.
  - color Op = Oid × Mode; Defined operations on objects.
  - color Roles = { r₁, ..., rₘ };
  - color R_tok, R_p = 2Roles; Roles of tokens and places, respectively.
  - color Auth = “authorized” | “unauthorized”;
  - color Token = 6-tuple of < Uid, Sid, R_tok, R_p, Op, Auth >; Tokens represent access subjects.
  - color DSOD, SSOD = Roles×Roles; Dynamic and static SOD relationships.

- P is a finite set of places; Places represent access objects. If tokens are on a place then accesses are occurred between the subject and the objects.

- T is a finite set of transitions; A transition can be interpreted as the access enforcement functions which test subjects’ authorities by evaluating arc expressions.

- A is a finite set of arcs such that P ∩ A = T ∩ A = P ∩ T = φ;

- N is a node function defined from A → P × T ∪ T × P;
• C is a colour function defined from $P \rightarrow \Sigma$ such that $\forall \ p \in P$, $C(p) = \text{Token}$;

• G is a guard function defined from $T$ into expressions such that $\forall \ t \in T$: $[\text{Type}(G(t)) = \text{Boolean} \land \text{Type}(\text{Var}(G(t))) \subseteq \Sigma]$, (where $\text{Type()}$ and $\text{Var()}$ return types and variables of arguments, respectively); The expressions in guard functions direct particular operations of a system.

• I is an initialization function defined from $P$ into closed expressions such that $\forall \ p \in P$: $[\text{Type}(I(p)) = C(p)_{\text{MS}}]$

• X is access matrix information; The entry of X is the tuple of $\langle r, SL, OL \rangle$ (where $r \in \text{Roles}$, $SL \subseteq \text{Uid}$, and $OL \subseteq \text{Op}$). Semantically, $SL$ includes the authorized users for the role, $r$. Acquiring $r$, subjects can access the operations which are included in $OL$.

• E is an arc expression function defined from $A$ into expressions such that $\forall \ a \in A$: $[\text{Type}(E(a)) = C(p(a))_{\text{MS}} \land \text{Type}(\text{Var}(E(a))) \subseteq \Sigma]$, (where $p(a)$ is the place of $N(a)$ and $A_{\text{MS}}$ denotes a multiset over $A$); An arc expression function is defined separately for input and output arcs of a transition. An arc expression has the value of $<\text{uid}_{\text{in}}, \text{session}_{\text{in}}, r_{-}\text{tok}_{\text{in}}, r_{-}\text{p}_{\text{in}}, \text{op}_{\text{in}}, \text{auth}_{\text{in}}>$ for an input arc and the value of $<\text{uid}_{\text{out}}, \text{session}_{\text{out}}, r_{-}\text{tok}_{\text{out}}, r_{-}\text{p}_{\text{out}}, \text{op}_{\text{out}}, \text{auth}_{\text{out}}>$ for an output arc, where the tuples of the value are type of Token). The values are evaluated based on the access matrix information X as follows:

$- \text{uid}_{\text{out}} = f_{\text{uid}}(\text{uid}_{\text{in}}^{1}, ..., \text{uid}_{\text{in}}^{n})$: $f_{\text{uid}}$ calculates the uid of output token
– session_out = f_{session}(session_{1_{in}}, ..., session_{n_{in}}): f_{session} calculates the session number of output token

– r_tok_out = f_{R=tok}(r_{tok_{1_{in}}}, ..., r_{tok_{n_{in}}}): f_{R=tok} assigns the union of input roles to the output. Thus, a subject does not lose its authority by an access.

– r_p_out = f_{R=p}(p, X): f_{R=p} returns the set of roles which is permitted to access the output place, p.

– op_out = f_{out}(p): f_{out} returns the operation related to the output place, p.

– auth_out = if sec_rcv \land sec_trans \land sec_snd is TRUE, then the value is “authorized”, otherwise “unauthorized”.

Where sec_rcv, sec_trans, sec_snd are defined for each transition t ∈ as follows:

• sec_rcv is TRUE iff for all p_{in}^i ∈ \bullet t and tok_{in}^j is on the place p_{in}^i, R(p_{in}^i) \cap R(tok_{in}^j) \neq \phi (where \bullet t is preset of t, R() returns the roles of a given argument).

It guarantees that the AEF is executed in safe state of the system.

• sec_trans is TRUE iff SOD conditions hold.

– for any t’s input token tok_{in}^i, tok_{in}^j, if (tok_{in}^i.uid = tok_{in}^j.uid), then (r_1, r_2) \notin SSOD (where tok_{in}^i.uid is the uid of tok_{in}^i and r_1 \in R(tok_{in}^i), r_2 \in R(tok_{in}^j)

– for any tok_{in}^i, tok_{in}^j, if (tok_{in}^i.session = tok_{in}^j.session), then (r_1, r_2) \notin DSOD (where tok_{in}^i.session is the session of tok_{in}^j).
- \text{sec\_snd} \text{ is } \text{TRUE} \text{ iff for all } p^i_{out} \in t\bullet \text{ and the emitted } tok^j_{out} \text{ of the place } p^i_{out},

\[ R(p^i_{out}) \cap R(tok^j_{out}) \neq \emptyset \text{ (where } t\bullet \text{ is postset of } t). \]

Finally, we can define security of a system by following steps.

- \textbf{secure transition:} for all reachable markings, if the value of \text{auth} field of the emitted token is “authorized”, the transition can fire safely. Subjects can access with relevant roles by the transition.

- \textbf{secure marking:} if the values of \text{auth} field of all tokens of markings are all “authorized”, the marking is secure.

- \textbf{secure net:} if all transition in a net is secure, the all reachable marking is secure, and the net is secure.

- \textbf{secure system:} we can constitute a corresponding net with a system. If the net is secure, we can say the system is secure.

It is noteworthy that CCPN model is different from information flow models such as SCPN. SCPN focused on where does the information flows, while CCPN access control model aims at judging permitted or not permitted actions of subjects. If there is one subject, SCPN presents a similar topology to the net of CCPN without Procedural Restrictions. Without the procedural information, the basic CCPN net consists of only one place and one transition. However, if there is another subject which has different label, we need to expand the SCPN topology to model the additional subject by adding the corresponding transition. In case of CCPN, the topology is unchanged because the subjects are represented by colored tokens.
The basic CCPN net consisting one place and one transition can describe also the current RBAC. The traditional access control information is described with types and expressions, not with network topology. The topology of CCPN is only expanded when PCs are added. The objectives and the generated nets of CCPN are totally different from those of SCPN in the aspect of semantics and structures of nets.

The proposed model is also different from the previous RBAC specifications. CCPN has adopted a state-transition approach and still has traditional access control information. It is a kind of hybrid specification which is distinguished from the previous access control models.

The auth field in CCPN is defined for the security evaluation. By checking its value during the verification process with the state-space generation, we can test the security of each transition, and eventually can evaluate the security of a system. On the other hand, we can associate the value with the firability condition of a transition. In that case, the liveness property is closely related to the security because the firability of transitions depends on role assignments. Thus, we can verify the correctness of the role assignments by checking liveness of each transition in PCs.

Based on the definition of binding and steps of CPN[64], we can extended the enable condition as follows.

**Definition 4.3.2 (Binding of a Transition)**: A binding of a transition $t$ is a function $b$ defined on $Vart$, such that:

- $\forall v \in Var(t): b(v) \in Type(v)$.
- $G(t) < b >$.

\[ - 102 - \]
By $B(t)$ we denote the set of all bindings for $t$.

**Definition 4.3.3 (step):**

- A **token element** is a pair $(p, c)$ where $p \in P$ and $c \in C(p)$. (the set of all token elements is denoted by $TE$)

- A **binding element** is a pair $(t, b)$ where $t \in T$ and $b \in B(t)$. (the set of all binding elements is denoted by $BE$)

A **marking** is a multi-set over $TE$ while a **step** is a non-empty and finite multi-set over $BE$. The **initial marking** $M_0$ is the marking which is obtained by evaluating the initialization expressions:

$$\forall (p, c) \in TE : M_0(p, c) = (I(p))(c).$$

The sets of all markings and steps are denoted by $M$ and $Y$, respectively.

**Definition 4.3.4 (Enabled):** A step $Y$ is enabled in a marking $M$ iff the following two property is satisfied:

- $\forall p \in P : \sum_{(t, b) \in Y} E(p, t) < b \leq M(p) \land Var(t).auth = authorized.$

**4.4 Analysis**

The proposed Coloured Petri Net formal model is suggested to specify an E-RBAC system and to help security administration of the system. In this section, we show application examples of the formal model.

An E-RBAC system specification can be tested by two main analysis methods: simulation and verification. The automatic tool for the Coloured Petri Net, CPN
Figure 4.6: A CPN modeling result (1): A system with PC examples.

Tools[66] support the two analysis. We show the analysis results of CPN Tools with an example configuration.

4.4.1 An Example Configuration

Let us assume that we have a system configuration as follows;

- \( \text{USERS} = \{u_1, ..., u_i\} \)
- \( \text{ROLES} = \{\text{SysAdmin}, \text{User}, r_1, ..., r_j\} \)
- \( \text{Objects} = \{\text{logfile, mail, prg, file}_1, ..., \text{file}_k\} \)
- \( \text{Modes} = \{\text{read, write, open, close, execute, link, unlink}\} \)
Behaviors = \{\text{ExecuteMailProgram, AccessLogFiles, } b_1, \ldots, b_n\}\]

We can model the system and the above configuration with the CCPN model. In addition, assume that the system have two procedural constraints; One positive procedural constraint that express former log file management operations and one negative procedural constraint derived from the previous sendmail daemon race condition attack (Refer the examples in Sect. 3.6).

In the Fig. 4.6, the modeling result of the system is represented. The figure depicts the common structure of CCPN representation of a system. In this overall diagram,
there are three types of operations are exist. First, operations, which are not controlled under the procedural constraints, are at the top part of the diagram. And there is a positive PC part, and negative PC part. As introduced previously, CPN support a hierarchical structure. Therefore, if there is a set of operations are under procedural constraints, we can express as a CCPN graph and add the graph as a subnet of this diagram. The positive and negative examples of Fig. 3.12 are presented as hierarchical subnets of Fig. 4.7, Fig. 4.8, respectively.
Figure 4.9: The simulation results of the positive procedural constraints which is related to the log file manipulation. The three possible execution sequences of (a) open-read-close

The diagrams are drawn by the CPN Tools[66]. The CPN Tools support convenient drawing of CPN and provide automatic property tests. With the modeling tool, we can verify the correctness of our security configuration. The CPN tool provides two types of analysis methods;

- Simulation
- Formal verification

4.4.2 Analysis by Simulation

The first simulation example is the previous positive PC example of log file manipulation executions. Fig. 4.9 - Fig. 4.11 shows the simulation results of three kinds of log
Figure 4.10: (b) open-write-close

Figure 4.11: (c) open-close
Figure 4.12: The simulation results of the negative procedural constraints which is derived from the sendmail daemon race condition attack. The detected and isolated attack trial is shown in (a) the negative PC subnet file management operations. The simulation tool informs the enabled transitions and the movements of tokens on the net. With given sets of operations, the results show that the subjects of the system itinerate the PC net successfully. In the itineration, the other combinations of the operations are not permitted.

The second simulation results are for the negative procedural constraints. The example procedure is the previously mentioned sendmail attack. Fig. 4.12 and Fig. 4.13 shows the system successfully detect and isolate the subject which executes a race condition exploit code. With the simulation results, we can find that the system detects and deny the race condition attacks successfully.

On the other hand, we can also find that there is a possible sequence of execution in our configuration. The possibility can be removed if the security officer modifies the
4.4.3 Analysis by Verification

The CPN has formal properties that can be defined and proven mathematically such as liveness, reachability, home state, reversability, and so on. Among the properties, the most important security related property in CCPN is liveness property. Liveness is the property of all of the transitions of a net can be firable (or be enabled).

With our previous negative PC example, we can verify the liveness property of the net formally. The result of the analysis, the net is live as Fig. 4.14 shows. It means, there is an attack sequence in our configuration make the termination transition can be enabled.

In this context, the security officer of a system can change the configuration to reduce the possibility of the attacks. In our example, we remove the permission of link

Figure 4.13: (b) overall diagram
Figure 4.14: The Result of Liveness Check of The Negative PC.

Figure 4.15: The Result of Liveness Check of The Negative PC with The Modified Configuration.
in the system for simplicity. Then, the result of the analysis is the existence of the dead transition as shown in Fig. 4.15. The dead of a transition means it will not be enabled forever. The dead transitions are detecting transition and terminating transitions of attacks. Therefore, with this system configuration, it is guaranteed that there is no possible attack sequence.

4.5 Summary

We reviewed various formal approaches in access controls. There are two main categories in formal methods: proof based methods and state machine based methods. Information flow models have been described with state machine based methods, but Role Based Access Control models have been specified with proof based approach. The E-RBAC model introduced in the Chap. 3 is also specified with proof based approach extending the standard RBAC specification. However, those specifications are not appropriate to describe execution sequences of the constrained E-RBAC model.

We proposed an interpretation and an extension of CPN as a formal model for the extended RBAC system with Procedural Restrictions. The proposed model, CCPN specifies the execution flow as well as it includes the traditional concept of access matrix. It is distinguished from the traditional information flow models or access matrix models. It is a kind of hybrid method in terms of it describes two access conditions that are specified with the different formal approaches.

In addition, we presented analysis examples of the example system which is modeled by the proposed CCPN. The simulation and formal verification analysis help to test
security-related properties of a system. It helps the security administration of E-RBAC systems.
Chapter 5

Implementation and Performance Evaluation

5.1 Introduction

In this chapter, we introduce the implementation example of our extended access control. Trusted operating systems have been implemented based on various kernels [17, 18, 19, 15, 21, 20]. Operating system kernels are categorized as micro-kernel approaches and integrated-kernel approaches. The current active researches are based on integrated-kernels, especially the Linux kernel. The Linux kernel has been updated and improved constantly. Moreover the kernel source is opened in public. Therefore, the current SELinux project, which is the most active trusted operating system kernel development project, is based on the Linux kernel. Considering the advantages of the Linux kernel, our implementation is also based on the Linux kernel.

We implemented E-RBAC on embedded environments. Embedded systems are important elements for massively distributed systems such as ubiquitous computing and pervasive computing environments, sensor networks, and so on. There are not enough resources to execute a large application such as intrusion detection systems and private firewalls in an embedded system. Therefore, the advantage of E-RBAC is outstanding in embedded environments E-RBAC provides such security services at the
kernel level as well as traditional access control services. We show the implementation of the E-RBAC system, and its attack detection results.

In addition, we evaluate performance overhead of the E-RBAC implementation. The E-RBAC is expected to have overhead due to the considerations of additional constraints. We measure how much performance overheads are produced in the system.

5.2 Implementation

There was no even a de facto standard for the implementation, although the core function of access control and the security policy of the implementations are not so different. However, there was a remarkable movement in the field of trusted operating systems based on monolithic Linux kernels. The Linux Security Module (LSM) [48] was developed to provide a bottleneck interface for access controls. Various access control policies can be enforced by the policies are implemented as the Loadable Kernel Modules. LSM is accepted officially as one of the security mechanisms of the Linux kernel from the kernel version 2.6. Additionally, SELinux, the most representative security kernel based on Linux, adopted it as its core framework of access controls. Therefore, LSM consolidates itself as the standard access control framework in Linux systems.

However, the LSM approach is not so reasonable for our implementation, because the LSM is too heavy mechanism to be ported into our tiny system. We implement our system on an embedded board, IFC-ETK100 [67], using the se3208 32-bit EISC processor [68]. The se3208 processor is a 16-bit processor actually, but it emulates
32-bit operations internally. Since there is no Memory Management Unit (MMU), the operating system is uClinux version 2.4.19 that can support the environment without MMU. The target embedded system has limitations in computational power and functions being compared to the Linux systems on desktop machines.

Many functions defined in LSM are not necessary in the embedded target. Some technologies in LSM are not supported in the system such as Loadable Kernel Module (LKM). Therefore, it can be an overhead to adopt the LSM architecture into such lightweight systems. Moreover, the LSM approach is not proven as the best or the most efficient solution for trusted operating system development [69].

Therefore, we implement our access control structure by directly modifying kernel functions of the system instead of taking the LSM architecture. At first, we simply add the field of a permission vector to the data structures of process and files. The permission vectors are calculated in terms of permitted roles. The relation between processes, roles, behaviors, and permissions is also evaluated as E-RBAC model, and the set of allowed permissions for the roles is calculated finally. Access control decision functions (ADF) decide the legality of each access comparing the roles of a process and a file. They are implemented as kernel functions. Their functions are not so different from those of the other trusted operating system implementations.

In addition, we add fields to trace the current state of the process in term of the CCPN model. Each process has its state information, and the state is changed by the execution of operations of the process. ADF calculates the next state according to the current state and the action that the process wants to execute. All processes itinerate
CCPN by executing operations. From the traces, we can investigate the execution sequence of operations of each access subject.

Fig. 5.1 shows an execution result of a race condition attack and its detection. The exploit program spawns two processes; one executes a sendmail process and the other repeats linking and unlinking to redirect the temporary file. The result shows that our implementation detects the attack successfully.

5.3 Performance Evaluation

For the performance evaluation of our simple implementation, we will execute three programs. The first two testing programs are a simple execution program and a copy program, respectively. The simple execution program executes the program that prints a short sentence in the LCD panel of the embedded system. The copy program makes copies of a 512-byte file. No procedural constraint is applied to the two programs. The third program is a log file read program that is executed under a procedural constraint. The state-transition graph of the action consists of three sequential states.
Figure 5.2: Comparison of the execution time between the original kernel and the modified kernel using E-RBAC from (a) a simple execution program, (b) a copy program, and (c) a log read program.

Figure 5.3: The overheads
Fig. 5.2(a) shows the execution time of the simple execution program and Fig. 5.2(b) shows the results of the copy program. Each program is executed repeatedly from 100 times to 1000 times, and the execution time is measured by using the wait3() function. As shown in the figures, without procedural constraints, there is no significant overhead in the E-RBAC system compared to the original kernel.

On the other hand, Fig. 5.2 (c) shows the different execution times between original kernel and our implementation. With procedural constraints, there is an overhead caused from the state tracing. It is shown from Fig. 5.3 that the overhead linearly increases as the repetition increases and the average of the overhead is about 7 milliseconds which corresponds to 10 percent increase in overhead. The overhead is almost equal to or larger than the overheads of SELinux[20]. However, it is not larger than the overhead of an application level intrusion detection solution of snort[70], when the overhead of operations in snort is measured in a firewall-applied environment.

5.4 The verification approach for implementations

Formal methods also have been introduced to test the correctness of the implementations of access control framework. In this section, we briefly introduce the survey result for the correctness test for the implementations.

As one of the research results of Linux Security Module[48] (LSM), they tried to confirm the access control functions are appropriately inserted to the kernel. LSM is a kind of common framework for access controls, and it has adopted into the Linux Kernels. Using a manual and partly automatic approach, they tried to find an access
which is not mediated by the access control mechanism [49, 50].

LSM (Linux Security Module) project have aimed at the framework development which can enforce various authorization mechanisms and policy. Based on the LSM framework, developers can build their own authorization modules and users can choose efficient and necessary parts among provided authorization modules. LSM provides such flexibility and convenience in authorization.

Since LSM module was adopted into the Linux kernel as a basic authorization module by Linus Torvalds, LSM have been a *de facto* standard of the authorization implementations for Linux. Besides, one of the representative examples of trusted operating system, SELinux also adopted LSM as their authorization framework.

It is what LSM does that inserting authorization hooks into the kernel. Not being located in system call bodies, the hooks are placed in kernel functions such as file system, memory management system, process management systems, and so on. All user processes’ calls pass the system call. Therefore, the syscall bodies can be the avoidable bottleneck points for access controls. However, LSM inserts the hooks into the several places of kernel functions instead, although it is easier to locate them in system calls. There are several reasons as follows.

First, in kernel functions, the hooks can be placed rather nearer to the access objects. It is the kernel functions which the objects are directly manipulated. Therefore, by locating hooks at the kernel functions, we can prevent the access objects from being replaced with the other objects during the access control process. In other words, it reduces the possibility of time-to-check-to-time-to-use (TOCTTOU)[23] attacks. The
TOCTTOU attacks exploit the time interval between the time of authorization and the time of operation invoked.

Second, access control system uses the object information with easy. In some cases, it is hard to access the information of the resource objects out of the kernel function. The hooks can access the information simply in the kernel function. The information does not need to be passed with complicated process such as argument passing.

Third, the authorization hooks in kernel functions help to increase security and performance.

Placing hooks in kernel functions gives some advantages; however we cannot guarantee whether the hooks mediate all of user processes’ access requests. Therefore, we need to confirm the coverage of the authorization hooks. It goes without saying that it is hard to be performed manually.

Edwards et al.[49] tried to check the correctness of the LSM authorization hook placements with the help of some automatic tools. They categorized the operations in a system as follows.

- security-sensitive operations: the operations influence the security of the system.

- controlled operations: the mediating interface which is the subset of the security-sensitive operations. They mediate accesses of the subset to other security-sensitive operations.

- authorization hooks: the LSM hooks.

- policy operations: the representatives of authorization hooks which are necessary
Figure 5.4: The comparison of the system call approach and the LSM approach.

for policy enforcements. They are the first controlled operation requiring the policy.

We can say that the arrangement of the authorization hooks is correct if it is guaranteed that authorization hooks control all security-sensitive operations. Authorization is the process of picking out allowable security-sensitive operations in a system, but it is comfortable to express such authorization policies in higher level. LSM authorizes rather policy operations than security-sensitive operations.

The comparison diagram of the system call mediation and the in-kernel mediation is in Fig. 5.4. If authorization hooks are in system call interface, policy operation...
and controlled operation are same. For example, the conceptual write operation is the syscall write in real. The policy is described in system call interface, and the system calls are the interface of the mediations. If the mediation is occurred in the kernel source, then the location of the mediation interface becomes unclear, policy operations and controlled operations are different. In this context, policy operations are the operations authorized by hooks and controlled operations mediate accesses to security-sensitive operations. All of operations in the kernel deal with sensitive data, and then controlled operation cannot be a form of interface function anymore. Policy is defined in the higher level of the set of controlled operation and policy operations are authorized controlled operations.

In the viewpoint of the verification, it cannot be grasped the relation between controlled operations and policy operations. For example, in case of the system call, we can verify the file open operation for the file write system call. However, in case of LSM, the authorization checks are executed at directory, link, and file accesses for the same operation. This kind of authorization reduces the possibility of TOCTTOU attack, but there are many work have to be done such as the verification of control operations and the policy operation for the control operations.

This is the basic idea of the placement verification of LSM authorization hooks. First, identity control operations and determine authorization requirements. Next, verify that authorization hooks for the necessary policy operations are authorized appropriately by the execution of the controlled operations.

For this verification, we need:
• a method to find control operations in kernel,

• the requirements of the control operations because control operations are executed at lower level than policy operations, and

• comparison of expected requirements and the invoke of LSM authorization hooks using automatic tools.

The idea of LSM analysis based on such an expectation that LSM authorization hooks are almost appropriate, so if there is an error it will not accord with the other part of the authorization mechanism. If a control operation has different authorization requirements for the same system call executions, then there is an error. Control operations are totally ordered for the influencing authorizations. All control operations in one system call have same authorization, and the functions of authorization are not affected by the other attributes of operations.

The assignments of the authorization functions are verified with following steps.

• CQUAL static analysis

• Type error analysis

• Generate causal requirements

The type-based static analysis tool CQUAL[51, 52] was used first. In CQUAL, user-defined type qualifiers can be used as the type of C language. For the verification, CQUAL type checked and unchecked are defined as the user-defined types, and the relation of checked and unchecked type qualifiers are defined as the input to the CQUAL.
CQUAL checks the violation of the type relation using inference rules. CQUAL reports the variables with type error and the control path to the type error. That is, the basic idea is that a variable with unchecked qualifier is prohibited from being used where checked variable is needed. It simulates the situation of variables are authorized before they are used in controlled operations.

For this analysis, CQUAL needs the source code having type qualifier. Linux kernel is very huge program. Therefore, it is performed by automatically that the process of giving type qualifiers to the kernel source. The automatic process of GCC analysis carries out the following tasks.

- The object to be controlled are marked as unchecked initially.
- All arguments used in controlled operations will be marked as checked.
- Authorized objects are upgraded as checked in authorizations.

By the static analysis, they found one vulnerability. The Fig. 5.5 shows the founded TOCTTOU vulnerability in the implementation. The filep variable was authorized in sys_fcntl, but new value is assigned and a new version of filep is got in fcntl_getlk. Therefore, user process replace the referred file pointer during the mapping of the file descriptor, then it is exploitable.

After some errors are eliminated by the CQUAL static analysis, type error analysis is performed by the experts. As their experimental results, the type error rate was about 12 percent for Virtual File System of the kernel 2.3.9. About 30 variables are used in controlled operations without being authorized. The type errors of the Virtual
File System are categorized as three types

- variables used for initialization and used only in secure functions
- inodes extracted from type checked dentry structure
- unknown type error

The first two errors are not attackable. The last error was checked manually by experts.

To generate causal requirements three tasks were performed:

- confirm vulnerability using given benchmarks and exploit programs
- confirm authorization error
- determine authorization requirements for controlled operations
A runtime analysis tool, Vali was developed by IBM and used to identify the authorization for objects within kernel events. The changed authorization means the kernel event is defined wrong or authorization function is not operated properly. Recalls of kernel events are defined by filter rule. Experts who have knowledge of kernel and LSM authorizations write the rules. The example of anomaly authorization is as follow[49]:

Although f_owner is accessed in the fcntl system call of inode field, the authorization for the set_owner was not called (See Fig. 5.6). However, the other flags of fcntl were come out with other authorization. The reason was authorizations use same fields. It can bring errors and the error can be found if we write appropriate filter for other changed forms. If filter is written once, Vali generates object authorizations, and uses it for the creation of anomaly authorizations.
5.5 Summary

In this chapter, we introduced an E-RBAC implementation and discussed its performance evaluation results. The extended access control efficiently limits attack trials consisting of allowed operations. For example, the implementation detected the send-mail race condition attack successfully.

Also, the performance overhead was evaluated with three simple programs. When operations were executed without procedural constraints, there was no significant overhead. On the contrary, the performance overhead was observed with procedural constraints. However, the overhead was not larger than the current intrusion detection solution. Considering that E-RBAC provides more fundamental security mechanisms which cannot be by-passed, this amount of overhead can be a bearable overhead. Moreover, usually in lightweight systems such as our embedded environment, it is impossible to execute heavy programs such as intrusion detection systems.
Chapter 6

Conclusions

The main security service of current security kernels has a functional limitation. They are incapable of protecting against the attacks that consist of ordinary operations. The incapability is originated from the pattern of current access control process. While making access decisions, access controls have not consider the associated information between accesses. The access rights on the operations have been checked separately.

The problem of current access control is solved if the security kernel control accesses consider the associated access information, that is the execution sequences of operations. In this thesis, we proposed a new access control scheme considering the sequence information in accesses. The access control scheme extends the abstraction concept of RBAC to give constraints to the set of permissions. Without extended access control, security kernels cannot detect and deny the attacks that comprise ordinary operations.

We also propose the formal specification method for the extended access control. The specification method is based on the Coloured Petri Net formalization. The proposed formal method can represent not only the traditional permitted relation between subjects and objects, but also execution sequence information. By the proposed formal model, we can specify current state of the system, and the various security related-properties of current system can be verified using automatic tools such as CPN Tools. Without the formal method, we should tests correctness of current security
configuration of the system.

In addition, we implemented the proposed access control scheme in an embedded environment. The objectives of the implementation are as follows. First, the guideline is given for the implementation of the proposed scheme. Second, the performance overhead is evaluated. The extended scheme is expected to have performance overheads, because the scheme considers additional constraints as well as the traditional access matrix information. To consider the constraints, we should trace subjects’ states, and then the performance overheads will be generated. We measured the performance overhead of our embedded implementation, and the extended scheme would not have a large overhead as compared to the former security solutions. The overhead was a bearable overhead, therefore our extended scheme is reasonable. Without a significant overhead, we can provide more fundamental security mechanism at the kernel-level than the application level solutions. We can also specify the access control model and a kind of intrusion detection mechanism using a unique formal model. This will make it easy to specify the security mechanisms, and then we can verify security-related features from the organized system.

To sum up, we got following three major achievements from this research:

- The extended RBAC, which enhances the functionality of traditional access controls
- The CCPN model can verify CCPN and help administration based on hybrid formalization
- The simple framework of trusted OS implementation.
References


25. 8lgm, Advisory 20, UNIX.SunOS, sendmail V5.1, Aug 1995.


42. R. Sekar, T. F. Bowen, and M. E. Segal, “On Preventing Intrusions by Process


48. LSM: http://lsm.immunix.org/


51. CQUAL: http://www.cs.berkeley.edu/~jfoster/cqual


55. Z: http://archive.comlab.ox.ac.uk/z.html

56. Alloy: http://sdg.lcs.mit.edu/alloy/


66. CPN Tools: http://wiki.daimi.au.dk/cpntools/
67. InterFC: http://www.interfc.co.kr

68. Advanced Digital Chips Inc.: http://www.adc.co.kr

69. gresecurity: http://www.grsecurity.net/lsm.php

70. Snort: http://www.snort.org
Curriculum Vitae

Name : Wook Shin
Birth Date : April 28, 1975
Birth Place : Seoul, Korea
Permanent Address : Dept. of Information and Communications, Gwangju Institute of Science and Technology

Education

1994.3–1998.2 Computer Engineering, Dongguk University (B.S.)
1998.3–2000.2 Information and Communications, Gwangju Institute of Science and Technology. (M.S.)
2000.3–2005.2 Information and Communications, Gwangju Institute of Science and Technology. (Ph.D.)

Honors and Awards

1998–2000 Government support, Ministry of Science and Technology
2000–present Government support, Ministry of Science and Technology

Patents

2002 Extended-BLP Model Based Access Control System, 0454231, Dong-Ik Lee, Jung-Min Kang, Wook Shin, Chun-Gu Park
Publications

International Journals


Lecture Notes in Computer Science


International Conference


Domestic Journals


Japanese Conference


**Domestic Conference (Korean)**


2. 신욱, 이동익, 역할-행위 기반 접근통제 한국정보처리학회 추계 학술발표논문집, Vol. 6, No. 2, pp. 57-63, 1999. (Best paper)


7. 신욱, 이동익, 김형찬, 강정민, 이진석, 보안 운영체제를 위한 확장된 역할기반


15. 박춘구, 신욱, 강정민, 이동익, 객체의 안전한 보안등급의 하강을 위한 접근통

16. 고기웅, 신욱, 이동익, 시스템 효율을 이용한 침입예상 데이터베이스 기반 침

17. 김형찬, 신욱, 이동익, 편재 컴퓨팅 기반에서의 보안 문제, 한국정보처리학회 추

18. 김형찬, 신욱, 이동익, 협업을 위한 역할기반 접근통제 모델, 한국정보과학회 추

19. 김형찬, 신욱, 이동익, 배타적 인증관리 도메인을 위한 역할기반 접근통제 모