情報保護學會論文誌
第16卷 第2號, 2006. 4

유연한 접근통제를 제공하는 보안 운영체제를 위한 접근통제 보안구조*

김 정 순,[1†] 김 민 수[2], 노 봉 남[3‡]
[1]전남대학교 전산학과, [2]목포대학교 정보보호학과, [3]전남대학교 전자컴퓨터정보통신공학부

An Access Control Security Architecture for Secure Operating System supporting Flexible Access Control*

Jung-Sun Kim,[1†] Minsoo Kim[2], Bong-Nam Noh[3‡]
[1]Dept. of Computer Science, Chonnam National University,
[2]Dept. of Information Security, Mokpo National University,
[3]School of Electronics, Computer and Information Engineering,
Chonnam National University

요 약

본 논문에서는 유연한 접근통제를 제공하는 보안 운영체제에 적합한 새로운 접근통제 보안구조를 제안한다. 제안된 보안구조는 가상 접근통제 시스템을 추가하여 다양한 접근통제모델을 보안 운영체제에 쉽게 적용할 있는 특성을 제공한다. 또한, 기존의 리눅스 시스템의 표준 접근통제의 단점을 극복하기 위해 설계되었으며, 유연하게 보안모델들을 조합할 수 있을 뿐만 아니라 동적으로도 보안모델들을 적용할 수 있다. 제안된 접근통제 보안구조는 접근통제집행부분과 접근통제결정부분, 보안제어부분으로 분리되고, 가상 접근통제 시스템에 의해 접근통제 모델들은 계층적 구조로 추상화된다. 그리고 다양한 접근통제 모델을 적용함으로써 발생하는 정책충돌의 개념과 해결방법을 제시한다.

ABSTRACT

In this paper, we propose a new access control security architecture for supporting flexibility in Secure Operating Systems. By adding virtual access control system layer to the proposed security architecture, various access control models such as MAC, DAC, and RBAC can be applied to Secure Operating Systems easily. The proposed security architecture is designed to overcome the problem of Linux system's base access control system. A policy manager can compose various security models flexibly and apply them to Operating Systems dynamically. Also, the proposed architecture is composed of 3 modules such as access control enforcement, access control decision, and security control. And access control models are abstracted to hierarchy structure by virtual access control system. And, we present the notation of policy conflict and its resolution method by applying various access control model.

Keywords : *Secure Operating Systems, Operating Systems, Access Control, Security Architecture*

접수일: 2005년 11월 11일; 채택일: 2006년 3월 15일
* 본 연구는 정보통신부 대학 IT 연구센터 육성, 지원사업의 연구결과로 수행되었습니다.

† 주저자, cybersun@lsrc.jnu.ac.kr
‡ 교신저자, bbong@jnu.ac.kr

Ⅰ. 서 론

최근 컴퓨터 기술의 급속한 발전으로 인해 가정이나 사무실에서 언제 어디서든 전 세계에 연결된 개인의 컴퓨터와 네트워크에 접근하여 정보를 이용할 수 있게 되었다. 이런 기술의 발전으로 인가되지 않은 다수의 사용자에게 민감한 데이터가 개방되거나, 공격에 무방비 상태로 노출되고 있다. 안전한 정보의 공유와 정보의 보호를 위해 방화벽, 침입탐지 시스템 및 암호화 메커니즘 등의 보안 기술이 개발되었고, 이들은 네트워크나 서버들의 정보를 보호하였다. 그렇지만 이러한 보안기술은 애플리케이션 수준에서 작동하기 때문에 애플리케이션 버그와 같은 잠재적인 보안 취약점과 내부자의 침입 및 권한 남용과 오용에 대한 대응이 어렵고, 시스템이 해킹 당하면 스스로를 보호하지 못하는 근본적인 한계를 가지고 있다. 이러한 문제를 해결하기 위해서는 보안운영체제와 같은 새로운 보안기술이 필요하다[1,2].

보안 운영체제는 컴퓨터 운영체제상에 내재된 보안상의 취약점으로 인하여 발생 가능한 불법행위로부터 시스템을 보호하기 위하여 기존의 운영체제 내에 인증과 암호화 등의 보안기능이 통합된 보안커널을 이식한 신뢰할 수 있는 전산환경(TCB: Trusted Computing Base)의 구현이다[13]. 보안 운영체제의 동작 기반을 제공하는 보안 커널은 참조모니터(Reference Monitor)의 구현이며, 접근통제 기능을 제공한다. 접근통제는 시스템 내에서 발생하는 접근이 적절한지를 판단하는 과정이다. 참조모니터는 접근 발생시, 접근주체와 접근객체로부터 접근결정이 필요한 정보를 추출하여 보안 규칙에 적합한지를 판단한다. 접근통제를 위한 주요 보안 정책으로는 시스템 보안관리자에 의해 부여된 사용자와 객체의 보안 등급에 의해 정보에 대한 접근을 제한하는 강제적 접근통제(MAC), 사용자의 신원에 근거를 두고 객체에 대한 접근허가를 결정하는 임의적 접근통제(DAC), 필요한 역할과 역할이 수행할 수 있는 연산을 보안 정책에 맞게 정의한 후 사용자들에게 역할을 할당하여 접근통제를 수행하는 역할기반 접근통제(RBAC)[3,8] 등이 있으며, 이는 접근통제 기법에 따라 보안 운영체제 구현을 분류하는 기준으로 사용된다[12]. 보안운영체제의 접근통제 보안 구조에 대한 연구는 접근 집행부분과 접근 결정부분으로 분리하여 접근통제의 적용의 효율성을 극대화시키는 방향으로 발전되어 왔다. 또한, 보안 운영체제의 접근통제 보안구조는 다양한 접근통제를 적용할 수 있는 접근통제의 유연성과 커널로부터 접근통제기능을 분리하는 접근통제의 커널 독립성의 특징을 갖는다.

본 논문에서는 접근통제의 유연성과 커널 독립성을 보장할 수 있도록 다양한 접근통제정책을 적용할 수 있는 가상 접근통제 시스템 개념을 물리적인 커널과 논리적인 접근통제정책 사이에 추가한 새로운 접근통제 보안구조를 제안한다. 제안된 보안구조는 기존의 알려진 접근통제정책뿐만 아니라 새로운 접근통제정책을 쉽게 보안커널에 쉽게 적용할 수 있으며, 동적으로 접근통제정책을 변경하거나 교체할 수 있는 장점을 제공한다. 또한 정책의 충돌 문제를 해결하는 방법과 리눅스 벤치마크 프로그램을 이용하여 제안된 보안구조의 성능 및 안전성 분석결과를 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 보안구조에 관련된 연구에 대해 살펴보고, 3장에서는 새로운 접근통제 보안구조에 대해 설명한다. 4장에서는 가상접근통제 시스템에 대하여 설명하며, 5장에서는 제안한 기법의 성능 및 안전성을 분석한다. 그리고 마지막 6장에서 결론을 맺는다.

Ⅱ. 관련연구

접근통제를 위한 보안구조에 대한 연구는 참조모니터를 중심으로 오래전부터 연구되어 왔으며 많은 보안 운영체제에 적용되었다. 최근에 연구된 보안구조로는 RSBAC의 GFAC 보안구조[10,11], SELinux의 Flask 보안구조[6], Medusa 99의 ZP Security Framework 보안구조[16] 등이 있으며, 접근통제집행 부분과 접근통제 결정 부분을 분리하는 구조적 특징을 보인다. 본 논문에서는 유럽에서 많은 사용자와 커뮤니티를 확보하고 있는 RSBAC과 리눅스 커널 2.6 버전에 포함되어 배포되는 SELinux에 사용되는 보안 구조에 대해 간단히 설명한다.

1. GFAC 보안구조

GFAC(Generalized Framework for Access Control) 보안구조는 ISO에서 정의된 WDACF(Working Draft on Access Control Framework)의 개념을 이용하여 LaPadula에 의해 만들어졌다[10,11].

GFAC 보안구조의 모든 접근통제는 기초적인 개념들의 집합에 기반하고 있다. GFAC 보안구조는 그림 1과 같이 판정모듈인 ADF(Access Control Deci-sion Facility)와 접근통제 집행모듈인 AEF (Access Control Enforcement Facility)로 분리하여 보안정책에 독립적인 구조적 특징을 갖는다.

주체의 객체로의 접근은 AEF에 의해 모니터되고 AEF는 ADF에게 접근에 대한 허용 여부를 요청하게 된다. ADF는 주체 및 객체에 정의된 접근통제 규칙을 이용하여 결정을 내리고 그 결과를 AEF에게 회신한다. AEF는 ADF의 판정결과를 바탕으로 객체에 직접 적용하여 접근통제를 집행한다. ACI (Access Control Information)는 주체, 프로세스, 객체에 대한 정보를 보관하며, ACR(Access Control Rules)은 IF-THEN으로 이루어진 논리적 상태들의 집합을 보관한다. GFAC 보안구조를 기반으로 개발된 보안운영체제로는 RSBAC(Rule Set Based Access Control)이 있다 [4,7].

2. Flask 보안구조

Flask(Fluke Advanced Security Kernel)는 유타(Utah)대학의 Flux 프로젝트의 결과인 Fluke 마이크로 커널에 보안 기능을 강화한 보안구조이다. DARPA(Defence Advanced Research Project Agency)의 지원을 받으며 유타대학교의 Flux팀과 미국 국방성(DoD: Department of Defence)의 협력으로 개발되었다 [6].

기본적인 Flask 구조는 그림 2와 같이 클라이언트 서버관계로 표현되는 객체관리자(Object Manager)와 보안서버(Security Server)로 구성된다. 객체관리자는 GFAC의 AEF와 같이 주체와 객체 사이에서의 보안 정책을 집행하는 부분이며 보안서버(Security Server)는 GFAC의 ADF와 같이 해당 주체와 객체 사이의 접근 허용 여부를 판단하는 부분이다.

Flask의 목표는 다양한 보안 정책을 수용할 수 있는 유연성(flexibility)과 투명성, 모듈성, 계층적 보안을 설계원칙으로 하고 있다. Flask 보안구조를 기반으로 한 보안운영체제로는 미국 NSA(National Security Agency) 주도로 이루어진 SELinux (Security-Enhanced Linux)가 있다 [5,14,15].

Ⅲ. 가상접근통제 시스템을 이용한 접근통제 보안구조

그이 장에서는 접근통제의 유연성과 커널 독립성을 보장할 수 있도록 가상 접근통제 시스템 개념을 추가한 새로운 접근통제 보안 구조를 제안한다. 제안된 구조는 접근통제집행부분과 접근통제결정부분, 보안제어부분으로 구성된다. 접근결정 이후 발생할 수 있는 문제를 해결하기 위해 보안제어부분을 새롭게 추가하였다. 또한, 가상 접근통제 시스템을 제안한 보안구조에 적용함으로써 다양한 접근통제정책을 지원할 수 있고, 동적으로 접근통제 정책을 변경하거나 수정할 수 있으며, 접근통제를 커널소스로부터 분리할 수 있는 장점을 제공한다.

1. 구성요소

제안된 접근통제보안구조의 접근통제 서비스는 행위의 주체(Subject), 행위의 대상인 객체(Object),

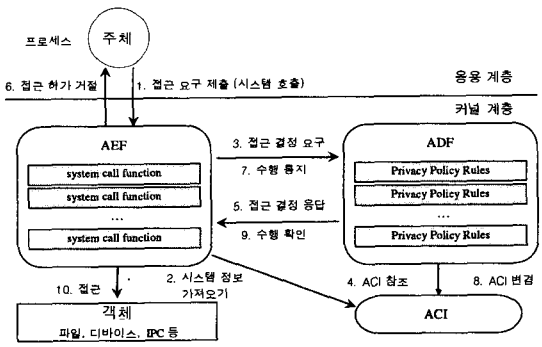


그림 1. GFAC의 보안 구조도

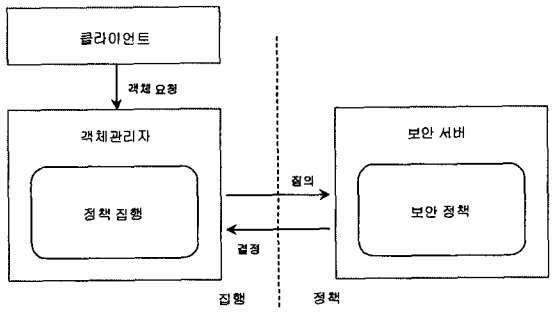


그림 2. Flask의 보안 구조도

그리고 주체가 객체에 행하는 행위(Action)의 기본 요소로 구성되며, 접근결정(Access Decision)은 이들 구성요소들의 관계로 표현된다.

1.1 접근주체 (Subject)

접근주체는 접근객체에 대해 허가된 행위를 수행하는 행위자로 제안된 보안구조에서는 사용자(User) 또는, 프로세스(Process)가 된다. 그러나 일반적인 유닉스/리눅스 환경에서 사용자는 시스템의 자원을 사용할 수 있는 허가받은 접근주체이지만 객체에 대한 직접적인 접근행위를 수행하지 않고, 프로세스를 통하여 객체에 대한 접근행위를 수행한다. 사용자와 프로세스는 밀접하게 연관되어 있으며, 유닉스계열 시스템 환경에서는 사용자 정보를 프로세스의 자료구조에 포함한다. 따라서, 프로세스를 사용자로부터 권한을 위임받은 대행자 또는 실질적인 접근주체로 정의한다.

1.2 접근객체 (Object)

유닉스 계열 시스템은 객체들을 대부분은 파일 형태로 표현하여 관리한다. 그러나, 접근통제 서비스를 위해 모든 객체들을 파일 형태로 관리하게 되면 접근통제 서비스를 위한 자료 구조 및 관리가 복잡하게 되는 문제점이 발생하므로 공통된 특성을 갖는 객체들을 클래스별로 분류하여 관리하면 보다 효율적인 접근통제 서비스를 제공할 수 있다. 따라서, 본 논문에서는 리눅스 시스템의 자원의 특성에 따라 표 1과 같이 파일객체, 프로세스객체, 네트워크 객체, 시스템객체 등 4개의 클래스로 분류하였고, 접근통제 서비스를 수행할 때 특별히 접근객체에 대한 접근권한 검사가 필요하지 않고 접근주체의 접근권한 검사가 필요한 접근통제 서비스나 앞의 클래스 분류에 속하지 않는 객체들을 관리하기 위해 미분류 클래스를 추가하였다. 또한, 각각의 접근객체 클래스들은 1개 이상의 하위클래스인 객체 타입들로 구성된다.

그리고, 메모리 객체는 리눅스 운영체제에서 가상메모리 방식으로 관리하므로 일반 사용자가 운영체제가 허용하지 않는 방법으로 다른 사용자의 메모리 영역을 접근할 수 없다. 즉, 운영체제가 직접 메모리관리를 하므로 운영체제의 기능을 제한하는 특별한 접근권한을 접근주체에게 부여할 필요성이 없기 때문에 본 논문에서는 메모리객체에 대한 접근권한은 파일 객체나, 프로세스 객체 등에서 간접적으로 관리한다. 예를 들면, 공유메모리를 사용하기 위해서는 운영체제가 커널 메모리 영역에 필요한 공간을 할당하게 되는데, 접근주체가 구체적으로 커널의 메모리 영역의 할당 범위를 직접 제어하는 것이 아니라 간접적으로 해당 시스템 콜인 shmget()을 통하여 접근주체가 공유메모리를 할당 받을 수 있게 한다.

표 1. 접근객체의 분류

클래스	유형	설명
File	FILE	파일객체
	DIR	디렉토리객체
	FILE SYSTEM	파일시스템객체
	DEVICE	디바이스객체
Process	IPC	IPC객체 (세마포어, 메시지, 공유메모리)
	PROCESS	프로세스객체
	SIGNAL	시그널객체
Network	NIC	네트워크인터페이스(NIC)객체
	SOCKET	소켓객체
	IP	IP 주소객체
System	SYSTEM	시스템 및 관리를 위한 객체
	USER	사용자객체
Unclassified	UNCLASSIFIED	분류되지 않는 객체

1.3 접근행위 (Action)

접근행위는 주체가 객체에 대해 행하는 일련의 구체적이거나 논리적인 행동으로 접근객체에 허용된 연산들이다. 접근행위에 대한 정의와 분류는 리눅스 시스템 콜을 바탕으로 접근 객체 클래스와 타입별로 정의하였다. 각 객체클래스는 타입별로 중복된 접근행위를 가질 수 있으며 객체에 따라 실질적인 접근행위결과는 다를 수 있다.

예를 들어 File 클래스의 FILE 타입 객체에 대한 create연산은 파일을 생성하며, Network 객체의 SOCKET타입의 객체에 대한 create 연산은 소켓을 생성한다. 동일한 create연산에 대해 객체유형별로 접근행위결과는 다르게 나타난다. 표 2는 제안된 보안구조에서 사용되는 접근객체에 허용된 접근행위들이다. 이러한 접근행위는 각각 시스템 콜이나 제안된 보안구조에서 제공하는 함수와 매핑된다.

표 2. 접근객체에 허용된 접근행위

클래스	유형	행위(Action)
File	FILE	create, delete, open, close, read, write, execute, truncate, rename, lock/unlock, link/unlink, mmap, munmap, stat, etc,
	DIR	create, delete, open, close, read, rename, mknod, mount, unmount, link/unlink, etc.
	FILE SYSTEM	register, unregister, mount, unmount, etc,
	DEVICE	open, close, read, write, ioctl, etc.
Process	IPC	msgctl, msgget, msgrcv, msgsnd, semop, semget, semctl, shmat, shmdt, shmget, etc.
	PROCESS	fork, execute, wait, kill, ptrace, chdir, chroot, sleep, etc.
	SIGNAL	kill, alarm, pause, sigaction, suspend, etc.
Network	NIC	create, close, read, write, etc.
	SOCKET	create, close, read, write, bind, listen, accept, connect, get/set socket options, etc.
	IP	accept, bind, connect read, write, etc
System	SYSTEM	create/delete module, reboot, shutdown, sysctl, syslog, quotactl, swap on/off, getrlimit, getusage, settime, etc.
	USER	add, delete, switch, assign/revoke role, etc.
Unclassified	UNCLASSIFIED	allow, deny, etc.

1.4 접근결정 (Access Decision)

접근결정은 접근주체, 접근객체, 접근행위의 3개의 원소로 구성된 접근요청에 대한 접근허용 및 접근거부를 결정하는 것을 말하며, 제안된 접근통제 보안 구조의 보안관리자 모듈에서 접근결정을 수행한다. 예를 들면, 사용자 U1이 응용프로그램인 vi를 사용하여 파일객체 /etc/passwd을 읽을 때 운영체제에서는 그림 3과 같은 접근통제서비스가 수행되며, 해당 시스템의 접근통제정책에 따라 접근요청에 대한 접근결정이 결정된다.

접근결정은 4장에서 설명되는 가상접근통제시스템에 의해 처리되며, 가상접근통제시스템은 시스템에 커널 모듈로 적재된 접근통제모듈의 접근결정메커니즘을 실행하여 접근통제서비스가 정상적으로 이루어지도록 한다.

2. 기본구조

제안된 접근통제 보안구조는 그림 4와 같이 총 3가지의 기본 모듈로 구성되어 있다. 첫 번째는 접근통제요청 및 집행을 수행하는 보안대행자(Security Agent) 모듈이고, 두 번째는 접근통제 요청에 대한 접근판단을 하는 보안관리자(Security Manager) 모듈이다. 세 번째는 접근통제 결정 이후 발생할 수 있는 오류 및 정책 충돌에 대한 제어를 하는 보안제어중재자(Security Control Mediator)모듈로 구성된다.

제안된 보안구조에서 접근주체로부터 접근요청을 받아서 접근객체에게 접근행위를 집행하는 절차는 다음과 같다. 첫 번째 단계는 접근을 요청하는 과정으로 접근주체가 특정 접근객체에 대한 접근행위를 요청한다. 보안대행자 모듈이 접근주체로부터 받은 요청을 분석하여 접근결정에 필요한 정보(접근주체, 접근객체, 접근행위)를 생성하여 보안관리자 모듈에게 전달한다. 두 번째 단계는 접근결정을 하는 과정으로 보안관리자 모듈이 보안대행자로부터 받은 요청 정보를 분석하여 현재 설정중인 보안 정책과 비교하여 접근결정을 생성하여 보안제어중재자 모듈에게 전송한다. 세 번째 단계는 접근결정에 대한 제어를 수행하는 과정으로 보안제어중재자 모듈이 접근주체, 접근객체 및 접근행위에 대한 분석을 통하여 적절한 제어를 수행하고, 그 결과를 기록한 다음 제어정보를 생성하여 보안대행자에게 전송한다. 네 번째 과정은 접근을 집행하는 과정으로 보안대행자가 접근이 허용된 경우 접근객체에 접근을 집행하여 그

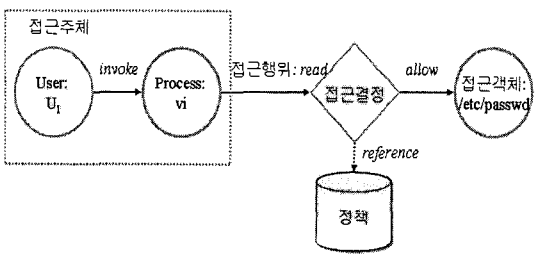


그림 3. 접근통제보안구조의 접근결정

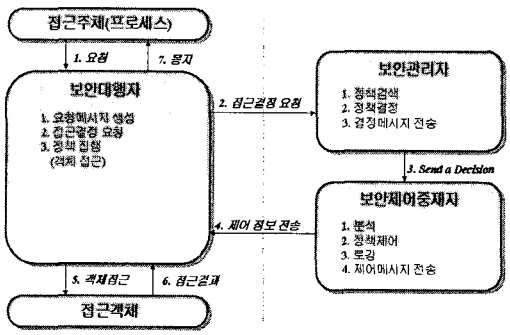


그림 4. 제안한 접근통제 보안구조도

결과를 접근주체에게 통보하고, 그렇지 않은 경우에는 접근주체에게 접근 거부 결과를 바로 통보한다.

2.1 보안대행자 모듈(SA: Security Agent)

보안대행자 모듈은 접근에 대한 결정을 보안관리자 모듈에게 요청하고 접근통제결과에 따라 접근객체에 대한 접근행위를 집행한다. 이 모듈은 접근주체는 누구이며, 접근주체가 접근하고자하는 접근객체는 무엇인지 그리고 접근주체가 목적으로 하는 접근객체에 어떠한 행위를 하는지를 식별한다.

보안대행자 모듈은 그림 5와 같이 컨텍스트 생성기(Context Maker), 접근요청메시지 전송기, 접근결정 및 제어정보 수신기로 구성된다. 보안대행자의 컨텍스트 생성기(Context Maker)는 접근주체의 요청을 분석하여 접근요청메시지(RM:Request Message)를 생성한다. 접근요청메시지는 접근주체와 접근객체, 접근행위의 쌍으로 구성되며 RM(Subject, Object, Action)의 형태로 표현한다. 접근주체는 접근행위를 요청한 프로세스의 사용자ID(uid) 또는 프로세스ID(pid)로 식별되며, 접근객체는 객체의 클래스와 타입에 따라 inode 번호, device 번호 등으로 식별한다. 접근행위는 표 2의 분류에 따라 고유번호를 부여하여 식별한다. 각각의 접근요청인 〈Subject, Object, Action〉은 식별 가능한 유일한 값으로 대응되도록 규정 되어 있으며, 이들로 생성된 RM을 통해서 시스템내의 접근주체가 접근객체에 어떤 접근행위를 하고자 하는지 명확하게 구분될 수 있다. 예를 들면 사용자 "U"가 임의의 프로세스를 실행하여 파일 "O"에 대해 "read"를 시도하면, 보안대행자는 요청에 따라 컨텍스트생성기를 통해 RM을 생성하여 보안관리자에게 접근결정을 요청한다. 이때 RM의 접근주체는 프로세스의 소유자(U)의 uid가 되고, 접근객체는 파일 "O"를 구분할 수 있는 유일한 값인 파일 "O"의 inode number와 device number가 되며, 접근행위는 "read"가 된다. 접근요청메시지 전송기는 생성된 접근요청메시지를 보안관리자 모듈에게 전송하고, 접근결정 및 제어정보 수신기는 보안제어중재자모듈로부터의 접근결정 및 제어정보를 포함하는 제어메시지 CM(Decision, Control)을 기다린다. 접근보안대행자는 보안제어중재자모듈로부터의 접근결정 및 제어정보를 수신하여 접근결정에 따라 접근이 허가인 경우는 객체에 접근행위를 집행하여 그 결과를 접근주체에게 통보하고, 접근이 허용되지 않는 경우는 허가거부결과를 접근주체에게 통보한다.

2.2 보안관리자 모듈(SM:Security Manager)

보안관리자 모듈은 보안대행자 모듈에서 요청한 RM메시지를 분석하여 설정된 시스템의 접근정책에 따라 접근요청에 대한 접근결정을 하고 그 결과를 보안제어중재자 모듈에게 전송한다. 이 모듈은 그림 6과 같이 가상 접근통제 시스템층(Virtual Access Control System Layer)과 접근결정메시지전송기의 두 부분으로 구성된다. 가상 접근통제 시스템에는 접근결정기와 정책검색기가 있으며, 추상적인 인터페이스만 제공하며 실질적인 기능은 보안정책별로 커널모듈로 구현하여 커널에 적재한다. 가상 접근통제 시스템에 대한 자세한 내용은 IV장에서 설명한다.

접근결정기(AccessDecision)는 보안대행자로부터 수신된 RM메시지를 분석하여 설정된 정책을 비교하여 보안대행자로부터의 접근요청에 대한 접근결정을 하여 접근결정메시지(DM:Decision Mess-

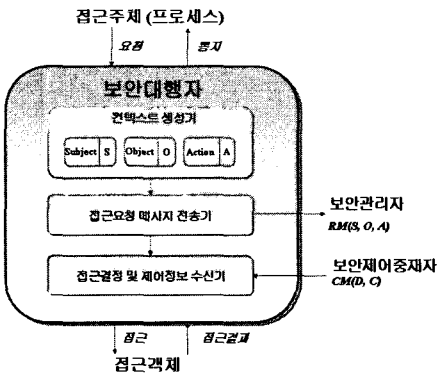


그림 5. 보안대행자 모듈 구조도

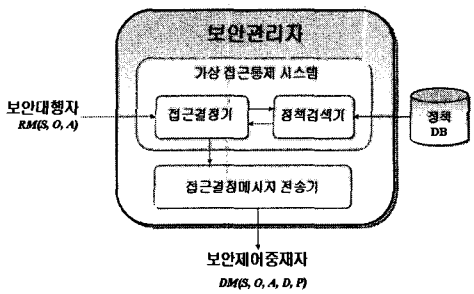


그림 6. 보안관리자 모듈 구조도

age)를 생성하는 모듈이며, 정책검색기(PolicySearch)는 정책이 저장된 정책데이타베이스로(Policy DB)부터 접근요청에 대한 정책이 정의되어 있는지 검색하여 그 결과를 접근결정기에 반환하는 모듈이다. 그리고 접근결정메시지 전송기(SendDM)는 접근결정기로부터 생성된 접근결정메시지를 접근제어중재자모듈에 전송하는 모듈이다. 접근결정메시지는 RM메시지와 접근결정정보, 그리고 정책정보로 구성되며 DM(Subject, Object, Action, Decision, Policy)으로 표현한다. 또한, 보안관리자 모듈은 내부적으로 RM메시지와 접근결정의 정보를 결정히스토리 저장소(Decision History Pool)에 저장하여 빈번한 접근요청들에 대한 접근결정을 빠르게 한다. 보안대행자로부터 접근요청메시지를 수신하면 보안관리자 모듈은 다음과 같은 처리과정을 수행한다. 먼저, 접근결정기는 결정히스토리에 접근요청에 대한 정보가 있는지 검색한다. 만약, 접근요청에 대한 접근결정이 저장되어 있으면 검색된 접근결정 정보를 이용하여 접근결정메시지를 생성하여 접근결정메시지전송기에 보낸다. 결정히스토리저장소에 해당 정보가 없으면 접근결정기는 정책검색기를 이용하여 정책데이타베이스로부터 해당되는 정책이 정의되어 있는지 검색하고, 검색된 정보를 이용하여 접근결정메시지를 생성하여 접근결정메시지전송기에 보낸다. 그리고, 정책데이터베이스에 정의되지 않는 정책의 요청인 경우에는 정책검색기가 정책검색 실패 오류를 생성하여 접근결정기에 보내고, 정책결정기는 이런 요청에 대해 접근거부 결정을 내린다. 그러나, 임의적 접근통제와 같이 사용자가 임의적으로 정책을 쉽게 변경할 수 있는 경우는 결정히스토리를 사용하지 않고 바로 정책검색기를 이용하여 접근결정메시지를 생성하여 접근결정메시지전송기에 보낸다. 마지막으로 접근결정메시지전송기는 접근결정기에서 생성된 접근결정메시지를 보안제어중재자모듈에 전송하여 접근통제작업이 계속 진행되도록 한다.

2.3 보안제어중재자 모듈(SCM: Security Control Mediator)

보안제어중재자 모듈은 내부자의 침입 및 권한 오·남용과 같은 접근결정 이후 발생할 수 있는 문제점을 해결하기 위한 모듈로 보안관리자 모듈에서 전달된 접근요청 정보, 접근결정 정보 및 정책 정보를 기반으로 접근주체, 접근객체 및 접근 행위에 대한 분석을 통하여 악의적이거나 비정상적인 접근요청이라고 판단될 경우 적절한 제어를 수행하고, 그 결과를 기록한 다음 제어정보를 생성하여 보안대행자에게 전달한다. 보안제어중재자 모듈은 그림 7과 같이 분석기(Analyzer), 제어기(Controller), 로거(Logger), 그리고 제어메시지전송기(SendCM)로 이루어진다.

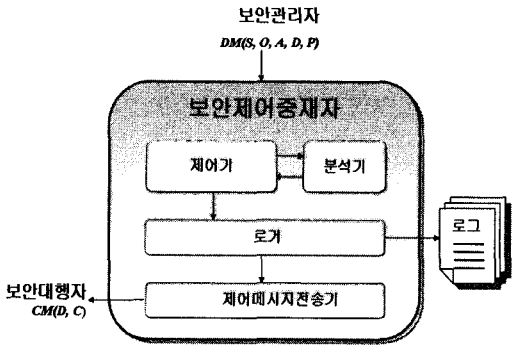


그림 7. 보안제어중재자 모듈 구조도

분석기는 각각의 접근주체와 접근객체 사이에서 형성될 수 있는 많은 관계 정보를 분석하여 정당성을 판단한다. 또한, 접근주체와 접근객체 그리고 접근행위를 잘 정의된 자료구조로 세분화하여 분석정보저장소(AIP: Analysis Information Pool)에 저장하고 저장된 정보를 이용하여 통계적 분석 및 오용탐지 그리고 비정상 탐지 등을 통하여 접근주체가 접근객체에게 행하는 접근행위에 정당성을 판단하게 된다. 제어기는 앞의 분석기에서 접근주체의 접근행위가 악의적이거나 비정상이라고 판단될 경우 또는 권한 남용과 오용인 경우로 판단된 경우, 접근을 차단하거나 서비스를 중지하는 등의 제어를 하며, 접근결정 정보와 제어정보를 이용하여 제어메시지(CM: Control Message)를 생성한다. 제어메시지는 보안대행자에게 전달되는 메시지로 CM(Decision, Control)으로 표현한다. 제어기 내부에는 악의적이거나 비정상적인 행위를 하는 접근주체나 행위의 대상인 접근객체에 대한 정보를 저장하는 블랙리스트저장소(BLP: Blacklist Pool)를 구성하여 보다 효율적으로 접근요청에 대한 제어를 수행한다. 로거는 관리자를 위한 것으로 로거의 설정에 따라 접근요청 시간, 접근요청위치, 접근요청정보, 접근결정정보, 제어정보 등을 기록한다. 마지막으로 제어메시지전송기는 제어기에서 생성된 제어메시지를 보안대행자에게 전송한다.

Ⅳ. 가상 접근통제 시스템(Virtual Access Control System)

가상 접근통제 시스템은 다양한 접근통제 메커니즘을 커널에 쉽게 적용하기 위한 방법으로 커널의 접근통제 메커니즘에 직접 접근하는 것이 아니라 하나의 계층적인 트리 구조로 통합하여 접근통제 메커니즘이 마치 하나인 것처럼 보이게 한다. 또한, 접근통제 메커니즘에서 사용되는 공통적인 접근통제 인터페이스를 제공하므로 적재 가능한 커널 모듈(Loadable Kernel Module)형태로 기존의 접근통제 메커니즘들을 구현하여 동적으로 다양한 접근통제 메커니즘을 커널에 적용할 수 있으며, 새로운 접근통제 메커니즘이나 개별 시스템 환경에 적합한 접근통제 메커니즘을 구현하여 쉽게 적용할 수 있다.

1. 가상 접근통제 시스템 구조

가상 접근통제 시스템은 그림 8과 같이 접근통제에 관계된 인터페이스들로만 구성된다. 즉, 이질적인 접근통제 메커니즘들의 동작에 종속되지 않도록 구체적인 구현부분은 제공하지 않고 공통적으로 사용될 수 있는 인터페이스들을 제공한다. 이 방법을 통하여 시스템에 다양한 접근통제 메커니즘들을 동적으로 적용할 수 있게 된다. 또한, 커널이 직접 접근통제 메커니즘을 관리할 필요가 없으므로 접근통제서비스를 커널과 독립적으로 구현할 수 있게 된다. 결과적으로 운용중인 시스템 커널의 직접적인 수정이나 변경 없이 다양한 접근통제 메커니즘들을 시스템에 적용할 수 있게 된다. 물론, 정책관리자는 이미 구현되었거나 직접 구현한 접근통제 메커니즘들을 시스템에 직접 적용해야 하며, 정책관리자의 실수나 부주의로 인한 시스템의 보안위험성을 증가시킬 수 있다. 그러나 이런 문제점은 모든 시스템들의 공통적으로 해당되는 잠재적 취약점이므로 제안된 구조만의 문제라 할 수 없다.

가상 접근통제 시스템과 접근통제 메커니즘들간의 동작 과정은 다음과 같다. 먼저, 정책관리자는 접근통제리스트(ACL: Access Control List), 강제적 접근통제(MAC: Mandatory Access Control), 임의적 접근통제(DAC: Discretionary Access Control), 역할기반 접근통제(RBAC: Role Base Access Control) 등의 접근통제 메커니즘들을 커널 모듈로 구현하여 커널에 적재하고 필요한 설정을 작업을 수행한다. 그러면, 커널은 접근통제 서비스 요청시 가상 접근통제 시스템의 정책결정기와 정책검색기 등의 인터페이스를 통하여 각각의 접근통제 메커니즘에 대응하는 접근결정기(Access Decision), 정책검색기(Policy Search)등의 구현된 메커니즘들을 사용하여 접근통제 서비스를 수행한다.

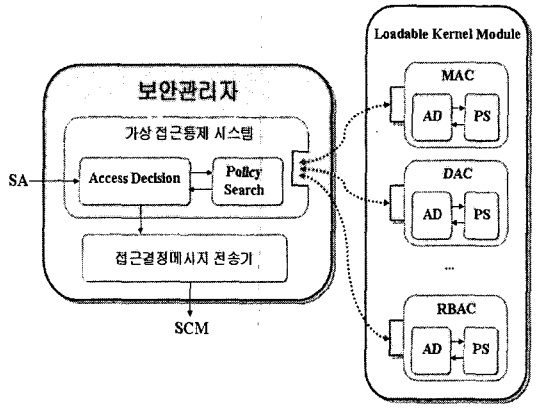


그림 8. 가상 접근통제 시스템 구조도

2. 가상 접근통제 시스템 인터페이스 및 자료구조

가상 접근통제 시스템은 표 3과 같이 접근통제 메커니즘 등록 및 해제 관리 부분, 접근통제 메커니즘 운용 관리 부분, 정책결정 및 검색 등을 관리하는 정책 관리 부분 등으로 구성된다.

접근통제 메커니즘의 등록 및 해제는 등록/해제 관리 인터페이스를 사용하고, 접근통제 메커니즘의 운용은 운용 관리 인터페이스를 사용한다. 커널 모듈로 구현될 접근통제 메커니즘은 access_decision, policy_search등과 같은 정책 관리 부분의 인터페이스들을 구현해야 한다. 또한, 가상 접근통제 시스템은 등록된 접근통제 메커니즘의 정보가 저장되는 vacs_policy_list 변수, 현재 적용중인 접근통제 메커니즘을 나타내는 vacs_current_policy변수, 등록된 접근통제 메커니즘의 수를 나타내는 vacs_policy_count변수, 정책리스트 자료구조(vacs_policy_list_t), 정책메커니즘 자료구조(vacs_policy_t), 정책 상태 자료구조(vacs_policy_state_t) 및 정책연산자 자료구조(vacs_policy_ops) 등이 있다. 정책연산자 자료구조는 표 3의 정책관리 부분에 해당하는 인터페이스들을 멤버변수로 포함한다.

표 3. 가상 접근통제 시스템 인터페이스

구분	API 함수명	API 설명
등록/해제 관리	register_vacs	가상 접근통제메커니즘을 커널에 등록한다.
	unregister_vacs	등록된 접근통제 메커니즘을 커널에서 해제한다.
	init_vacs	가상 접근통제 시스템의 자료구조를 초기화한다.
	get_vacs	등록된 접근통제 메커니즘에 대한 포인터를 얻는다.
운용 관리	set_vacs	사용할 접근통제 메커니즘을 설정한다.
	init_policy	접근통제 메커니즘의 자료 구조를 초기화한다.
	exit_policy	접근통제 메커니즘의 자료구조를 해제한다.
	access_decision	접근요청에 대한 접근결정을 수행한다.
정책 관리	policy_search	정책데이타베이스에서 해당 접근요청을 검색한다.
	get_sscontext	접근주체의 보안정보를 얻는다.
	get_oscontext	접근객체에의 보안정보를 얻는다.
	set_policy	시스템의 접근통제정책을 변경한다.
	get_policy	시스템에 설정된 접근통제정책을 얻는다.

3. 접근통제 메커니즘의 관리

접근통제 메커니즘이 구현된 원시파일을 컴파일하여 생성된 커널모듈을 insmod명령으로 커널에 적재하며, 커널에 적재된 접근통제 메커니즘 모듈은 rmmod명령을 사용하여 커널에서 해제한다. 접근통제 메커니즘 모듈은 적재될 때 커널에 자신을 등록하고, 제거될 때 자신을 해제한다. 각 접근통제 메커니즘의 초기화 루틴은 자신을 가상 접근통제 시스템에 등록하며, 초기화된 정보는 vacs_policy_list_t에 의해 표현된다. vacs_policy_list_t 자료구조에는 접근통제 메커니즘의 고유번호, 이름, 정책연산자 루틴에 대한 포인터 등이 저장된다. 그리고, set_policy, get_policy 인터페이스를 통하여 시스템의 접근통제 메커니즘을 변경하거나 얻어오며, 이때의 설정값은 vacs_current_policy 변수에 저장된다. 그림 9는 insmod명령을 사용하여 ACL모듈, MAC모듈, RBAC모듈 순서로 커널에 적재하고, set_policy를 통하여 RBAC 메커니즘으로 설정하였을 때의 vacs_policy_list_t 자료구조와 vacs_current_policy를 보여준다.

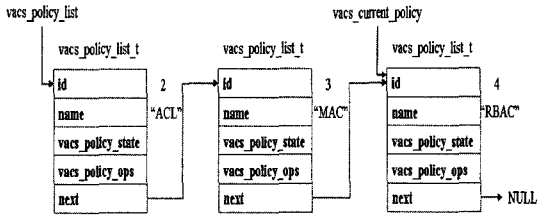


그림 9. 등록된 접근통제 메커니즘

4. 정책충돌 및 해결방법

가상 접근통제 시스템을 사용하면 동일한 시스템에 한 개 이상의 접근통제 메커니즘을 적용할 수 있다. 그러나 한 개 이상의 접근통제 메커니즘이 적용되면 접근통제 메커니즘간의 정책 결정에 대한 충돌이 발생한다. 즉, 정책충돌은 동일한 접근요청에 대해 접근통제 메커니즘들의 결정이 상반되었을 경우를 말한다. 이런 정책충돌은 내부정책충돌과 외부정책충돌의 두 가지 경우로 구분할 수 있다. 내부정책충돌은 한 정책 내에서 동일한 접근요청에 대해 여러 개의 상반된 정책들의 설정으로 인한 정책결정의 불일치를 의미한다. 반면에, 외부정책충돌은 서로 다른 정책들간의 동일한 접근요청에 대한 서로 다른 정책들의 설정에 의해 발생하는 정책결정의 불일치를 의미한다. 정책충돌의 위험성은 관리자의 정책설정 부주의 또는 정책검증 시스템의 오류에 의해 발생할 수 있으며, 정책 충돌이 발생하면 시스템의 접근통제 무결성을 보장하지 못하게 된다. 따라서 접근통제서비스는 정책 충돌에 대한 해결방법을 제공하여야 한다. 본 논문에서는 접근결정연산결과의 논리합, 논리곱, 가중값 함수를 사용하는 방법(우선순위 방법)으로 정책충돌의 문제의 해결방법을 제시한다. 이런 방법들은 내부정책충돌이나 외부정책충돌의 경우 모두에 사용가능하며, 본 논문에서는 외부정책충돌의 사례를 설명한다. 다음은 정책결정 방법을 위한 기본정의들이다.

〔정의 1〕 모든 접근통제 메커니즘들의 집합을 U, 시스템 S에 적용된 접근통제 메커니즘을 P라 한다. 시스템 S는 일반적으로 유닉스나 리눅스 또는 윈도우 운영체제를 의미한다. 본 논문에서는 리눅스 시스템을 예로 설명한다.

$U = \{x | x$는모든접근통제메커니즘$\}$,
$P = \{x \in U | x$는시스템S에적용된모든접근통제메커니즘$\}$

〔정의 2〕 접근요청들의 집합을 AR라 한다.
$AR = \{x \mid x$는 모든 접근요청들의 집합$\}$

〔정의 3〕 접근통제 메커니즘의 연산자 Access Decision을 다음과 같이 정의한다. 그리고, 접근결정값은 접근이 허가된 경우(allow)는 TRUE값이고 그렇지 않은 경우(deny)는 FALSE값을 갖는다.
(단, 기호 '→' 는 해당 객체의 구성요소를 참조하는 포인터 연산자와 같은 의미이다.)

$AccessDecsion(r)$ = 접근요청 r에 대한 접근결정함수 (단, $r \in AR$) 수식에서는 AD로 표현함

$P \rightarrow AccessDecsion(r) = \{x \mid x$는 접근요청 r에 대한 정책 P의 접근결정값$\}$ (단, $P \in U \wedge r \in AR$)

4.1 논리곱 방법

시스템 S에 적용된 모든 접근통제 메커니즘의 접근결정연산 결과의 논리곱을 정책결정결과로 사용하는 방법으로 적용된 모든 접근통제 메커니즘들의 접근결정연산이 allow인 경우에만 접근을 허용하고, 그렇지 않은 경우에는 접근을 허용하지 않는다. 시스템 S에 적용된 모든 접근통제 메커니즘의 접근결정연산들의 교집합을 구하기 위해 집합 P^s와 동일한 원소수를 갖는 첨수집합(index set) Γ을 정의한다. 집합 Γ의 각 원소 $\gamma \in \Gamma$에 대해 집합 $P_\gamma \rightarrow AccessDecision(r)$ 가 대응될 때, 모든 $P_\gamma \rightarrow AccessDecision(r)$ 를 모아 놓은 집합 $\{P_\gamma \rightarrow AccessDecision(r) \mid \gamma \in \Gamma\}$ 을 Γ에 의해 첨수가 매겨진 첨수집합족(indexed family of sets)이라고 이 집합족에 대해 교집합을 이용한 시스템 S의 정책결정방법 $AccessDecsion_s^*(r)$ 은 다음 식 (1)과 같이 표현된다.

$$\begin{aligned} AD_s^*(r) &= \{P_\gamma \rightarrow AD(r) \mid \gamma \in \Gamma\} \text{ 단, } P_\gamma \in P,\ \Gamma = \{1,2,\cdots,n = |P|\} \\ &= \bigcap_{\gamma \in \Gamma} P_\gamma \rightarrow AD(r) \\ &= \bigcap_{i=1}^{n} P_i \rightarrow AD(r) \\ &= P_1 \rightarrow AD(r) \bigcap P_2 \rightarrow AD(r) \bigcap \cdots \bigcap P_n \rightarrow AD(r) \\ &= \{x \mid x \in P_1 \rightarrow AD(r) \wedge x \in P_2 \rightarrow AD(r) \wedge \cdots \wedge P_n \rightarrow AD(r)\} \end{aligned} \quad (1)$$

식 (1)의 최종 결과에 의해 교집합을 이용한 시스템 S의 정책결정방법 은 접근요청 r에 대한 정책들의 충돌이 발생하였을 경우 S의 모든 접근통제 메커니즘들의 접근결정연산이 모두 allow인 경우에만 접근을 허용하게 된다. 이 방법은 가장 일반적으로 적용될 수 있는 방법으로 상반된 정책충돌이 발생할 경우 접근이 허용되지 않기 때문에 시스템의 무결성 및 기밀성을 보장할 수 있다. 그리고, RSBAC 보안운영체제는 ADF 모듈 구현에서 논리곱 방법을 사용하여 정책충돌 문제를 해결한다 [7].

4.2 논리합 방법

시스템 S에 적용된 모든 접근통제 메커니즘의 접근결정연산 결과의 논리합을 정책결정결과로 사용하는 방법으로 적용된 모든 접근통제 메커니즘들의 접근결정연산이 한 개 이상의 allow값이 있으면 접근을 허용한다. 앞 절의 방법과 유사하게 첨수 집합 Γ에 의해 첨수가 매겨진 첨수집합족에 대해 합집합을 이용한 시스템 S의 정책결정방법 $AccessDecsion_s^+(r)$ 은 다음 식 (2)와 같이 표현된다.

$$\begin{aligned} AD_s^+(r) &= \{P_\gamma \rightarrow AD(r) \mid \gamma \in \Gamma\} \text{ 단, } P_\gamma \in P,\ \Gamma = \{1,2,\cdots,n = |P|\} \\ &= \bigcup_{\gamma \in \Gamma} P_\gamma \rightarrow AD(r) \\ &= \bigcup_{i=1}^{n} P_i \rightarrow AD(r) \\ &= P_1 \rightarrow AD(r) \bigcup P_2 \rightarrow AD(r) \bigcup \cdots \bigcup P_n \rightarrow AD(r) \\ &= \{x \mid x \in P_1 \rightarrow AD(r) \vee x \in P_2 \rightarrow AD(r) \vee \cdots \vee P_n \rightarrow AD(r)\} \end{aligned} \quad (2)$$

식 (2)의 최종 결과에 의해 합집합을 이용한 시스템 S의 정책결정방법 $AccessDecsion_s^+(r)$ 은 접근요청 r에 대한 정책들의 충돌이 발생하였을 경우 S의 접근통제 메커니즘들의 접근결정연산이 모두 deny인 경우를 제외하고 접근을 허용하게 된다. 그러나 이 방법은 시스템의 정책 설정 오류가 운영체제에 그대로 반영되는 문제점이 있다. 예를 들면, P_{ACL} 과 P_{RBAC} 을 사용하는 시스템 S에서 접근요청 r에 대해 P_{ACL} 은 접근을 허용하게 정책이 설정되어 있고, P_{RBAC} 은 접근을 허용하지 않게 정책이 설정되어 있다면, 접근요청 r에 대한 $AccessDecsion_s^+(r)$ 은 $AD_s^+(r) = P_{ACL} \rightarrow AD(r) \vee P_{RBAC} \rightarrow AD(r) = allow \vee deny = allow$ 가 되어 접근을 허용하게 된다. 결국, 이 방법은 서로 상반된 접근결정을 갖는 정책들의 충돌이 발생하였을 경우에는 적합하지 않는 방법이다.

4.3 가중값 함수를 사용하는 방법

시스템 S에 적용된 모든 접근통제 메커니즘들에 가중값들을 부여하여 가중값이 가장 큰 접근통제메커니즘의 접근결정연산 결과를 시스템의 접근결정 값

으로 사용하는 방법이다. 시스템의 접근통제 메커니즘들에 대한 가중값 함수 w와 최대값 함수 MAX를 이용한 시스템 S의 접근결정방법 $AccessDecsion_s^w(r)$은 다음 식 (3)과 같이 표현된다.

$$AD_s^w(r) = P_s \rightarrow AD(r) \quad (3)$$
(단, P_s는 $MAX(w(P_1), w(P_2), \cdots, w(P_n))$가 되는 정책, $w: P \rightarrow R$)

식 (3)에 의해 가중값 함수를 이용한 정책결정방법 $AccessDecsion_s^w(r)$은 접근요청 r에 대한 정책들의 충돌이 발생하였을 경우 가중값이 가장 큰 P_s의 접근결정연산 결과에 따라 시스템 S의 접근통제 메커니즘이 결정된다. 예를 들어 P_{MAC}, P_{DAC}과 P_{RBAC}을 사용하는 시스템 S에서 접근요청 r에 대해 P_{MAC}과 P_{DAC}은 접근을 허용하도록 정책이 설정되어 있고, P_{RBAC}은 접근을 허용하지 않게 정책이 설정되어 있다. 그리고 정책의 가중값은 $w(P_{MAC}) = 0.3$ $w(P_{DAC}) = 0.2$, $w(P_{RBAC}) = 0.5$으로 설정되어 있다면, 접근요청 r에 대한 시스템 S의 접근결정은 $AccessDecsion_s^w(r) = P_{RBAC} \rightarrow AccessDecision(r) = deny$ (단, $P_s = P_{RBAC}$)가 되어 접근을 허용하지 않게 된다. 이 방법은 서로 상반된 접근결정을 갖는 정책들의 충돌이 발생하였을 경우 우선순위가 가장 높은 정책이 적용되게 하는 방법이다. 결국, 접근통제 메커니즘들에 우선순위를 부여하고 우선순위가 높은 것을 사용하는 방법과 같다. 또한, 논리합과 논리곱을 이용하는 방법은 가중값 함수를 사용하는 방법의 특수한 예로 모든 접근통제 메커니즘들의 가중값이 동일한 경우이며, 4.1과 4.2 에서 설명한 논리곱이나 논리합의 방법으로 정책충돌의 문제를 해결한다. 본 논문에서는 가중값 함수와 논리곱 방법을 사용하여 정책충돌의 문제를 해결한다.

```
decision_t access_decision(struct rm_t *rm) {
    decision_t decision = D_DENY;
    struct policy_t *policy = NULL;

    policy = MAX_POLICY(vacs_policy_list);
    if(policy != NULL)
        decision = policy->ops->access_decision(rm);

    return decision;
}
```

그림 11. access_decision함수

5. 가상 접근통제 시스템 구현

그림 10은 가상 접근통제 시스템의 주요자료구조인 vacs_policy_ops와 vacs_policy_t를 나타낸 것이다. 그리고 그림 11은 가상 접근통제시스템의 access_decision 인터페이스를 보여준다. 보안관리자 모듈에서 가상 접근통제시스템의 access_decision 인터페이스를 사용하여 등록된 접근통제 메커니즘의 access_decision 함수를 실행하여 접근결정을 한다. 가상 접근통제시스템의 access_decision 인터페이스는 가중값 함수를 사용하는 방법으로 구현하였으며, MAX_POLICY 함수는 가중값이 최대인 접근통제 메커니즘을 반환하는 함수이다.

그림 12는 MAC모듈의 접근결정함수인 access_decision함수의 구현 예를 보여준다. insmod 명령을 사용하여 LKM 모듈로 작성된 MAC모듈을 커널에 등록하면 가상 접근통제 시스템의 자료구조인 vacs_policy_list에 저장되고, MAC모듈의 access_decision함수는 vacs_policy_ops 자료구조인 ops 변수의 access_decision 멤버변수에 저장된다.

```
struct vacs_policy_ops {
    int         (*init_policy)(void);
    int         (*release_policy)(void);
    decision_t  (*access_decision)(struct rm_t *rm);
    ...
};

struct vacs_policy_t {
    int         id;
    char        name[MAX_POLICY_NAME];
    int         weight;
    struct vacs_policy_ops  *ops;
    ...
};
```

그림 10. 주요 자료구조

```
decision_t access_decision(struct rm_t *rm) {
    decision_t d = D_DENY;

    if (!verify(rm->s, rm->o, rm->a))
        return(D_UNDEFINED);
    if (dominates(rm->s, rm->o) &&
        action_map(rm->a, READ))
        return(D_GRANT);
    if (dominates(rm->o, rm->s) &&
        action_map(rm->a, WRITE))
        return(D_GRANT);
    return d;
}
```

그림 12. MAC 모듈의 access_decision 함수

가상 접근통제 시스템에서는 접근통제에 관련된 인터페이스만 제공하므로, MAC모듈에서 필요한 함수인 주체와 객체의 지배관계를 결정하는 dominates함수와 MAC에서 정의된 읽기, 쓰기와 같은 퍼미션을 실제 운영체제의 접근행위로 변환시키는 action_ map함수 등은 직접 구현하여야한다. 그리고, verify함수는 접근주체, 접근객체, 접근행위의 유효성을 검사하는 함수이다.

```
decision_t access_decision(struct rm_t *rm) {
    decision_t d = D_DENY;
    struct security_context_t ssc;
  . void *ret = NULL;

    if(!verify(rm->s, rm->o, rm->a))
        return(D_UNDEFINED);
    ssc.role = assigned_roles_to_subject(rm->s);
        ...
    ret = roles_has_permission(ssc.role, rm->o,
        rm->a);
    if (ret != NULL)
        return(D_GRANT);
    return d;
}
```

그림 13. RBAC 모듈의 access decision 함수

그림 13은 RBAC모듈의 접근결정함수인 access_decision함수의 구현 예를 보여준다. assigned_roles_ to_subject함수에서 접근주체에 할당된 역할을 찾고, roles_has_permission함수에서 이 역할에 해당 퍼미션(접근객체와 접근행위의 쌍)이 할당되어 있는지 검사하여 접근결정이 이루어진다.

6. 제안한 보안구조의 장단점

다양한 접근통제 정책을 지원하기 위해 GFAC 보안구조는 접근결정모듈인 ADF에 지원하는 모든 접근통제 메커니즘들을 구현하고, ACI에 관련 각각의 자료구조들을 생성해야 한다. 즉, 새로운 접근통제 정책을 지원할때마다 ADF에 해당하는 접근통제 메커니즘을 추가적으로 구현해야 하고, ACI에 관련 자료구조들을 추가해야 하므로, 접근통제서비스를 위한 커널의 자료구조와 메커니즘이 복잡하게 되는 문제점이 있다. 또한, 초기의 Flask 보안구조는 다양한 접근통제 정책을 지원하기 위해 보안서버들을 각각 구현해야 하는 구조적인 문제가 있었으며, 현재 SELinux에서는 이를 개선하여 RBAC, TE, MLS 개념을 결합한 형태의 확장 TE모델을 보안서버에 구현하여 다양한 접근통제 정책을 지원한다. 그러나, 새로운 접근통제 정책을 지원하기 위해서는 현재의 확장 TE모델에 새로운 개념을 추가하여 TE 모델과 관련 메커니즘들을 추가로 수정해야 하는 문제점이 있다.

제안한 보안구조는 정책집행부분과 정책결정부분을 분리하여 커널로부터 정책 결정 모듈을 분리시킬 수 있도록 하였고, 정책결정부분에 가상 접근통제 시스템을 추가하여 다양한 접근통제 모델을 지원하고 동적으로 정책 변경이 가능하도록 하였다. 그리고, 다양한 접근통제 정책들 사이에 발생하는 정책 충돌의 문제를 해결할 수 있다. 제안한 보안구조는 기존의 Flask 및 GFAC 보안구조와 상당히 유사하지만 표 4와 같은 장단점이 있다.

표 4. Flask, GFAC 및 제안한 보안구조 장단점 비교

구분	장점	단점
GFAC	- AEF와 ADF를 분리하여 접근 결정을 커널로부터 분리 - ADF에 접근통제 메커니즘들을 구현하여 다양한 접근통제 모델 지원	- 단순한 정책변경만 지원하고 동적인 접근통제 정책 변경 미흡 - 커널에 의존적인 데이터 구조 - 불완전한 접근결정모듈의 분리 - 정책결정 캐시기능의 부재
Flask	- 객체관리자와 보안서버를 분리하여 접근결정을 커널로부터 분리 - 정책결정 캐시기능 제공 - 다양한 접근통제 모델 지원	- 다양한 접근통제 모델을 지원하기 위해서는 정책서버 모듈을 각각의 정책들에 맞게 구현 - 동적인 접근통제 정책 변경 미흡
제안한 보안구조	- 보안대행자와 보안관리자를 분리하여 접근결정을 커널로부터 분리 - 다양한 접근통제 모델 지원 - 새로운 접근통제 모델을 쉽게 지원 - 동적인 접근통제 정책 변경 및 적용 가능 - 다양한 정책들간에 발생하는 정책 충돌 문제 해결 가능	- 접근결정 동작시 가상 접근통제 시스템 레이어를 통하여 구현된 정책 메커니즘을 수행하므로 오버헤드 발생 - 등록된 접근통제 메커니즘을 관리하는 자료구조가 연결리스트이므로 접근통제 메커니즘들이 많아질 경우 오버헤드 발생

V. 성능분석

이 장에서는 제안한 접근통제 보안구조를 리눅스 시스템에 설치하여 성능과 안전성 테스트를 수행한 실험결과를 설명한다. 성능분석은 lmbench [9,18] 를 이용한 마이크로벤치마크(microbenchmark)를 수행하였으며, 안전성분석은 LTP(Linux Test Pro-

ject) [17] 를 사용하였다. 실험방법은 제안한 접근통제보안구조를 설치하기 전과 설치후의 시스템의 성능과 안전성을 측정하여 비교하였다. 단, 실험대상인 제안된 보안구조는 보안구조와 가상접근통제 시스템만 것으로 특별한 보안정책이 적용되지 않는 경우이며, 제안된 보안구조+MAC은 제안된 보안구조에 MAC모듈을 구현하여 커널에 적재한 경우를 의미한다.

1. 실험환경

실험환경은 레드햇 리눅스(Fedora Core 3)버전을 인텔 Xeon(TM) CPU 3.06GHz, 1GB RAM 메모리, 512KB 캐시와 Fast Ethernet을 갖춘 시스템에 설치하고, 커널 2.6.13 버전으로 커널을 패치하여 구성하였다. 성능실험 항목은 프로세스와 메모리, 파일 및 네트워크간의 데이터 대역폭 및 지연율을 측정하였다. 그리고 안전성실험 항목은 시스템 콜과 기본커널모듈들의 안전성을 테스트한다. 실험대상은 리눅스 커널 2.6.13 버전을 설치한 시스템과 여기에 제안한 보안구조를 설치한 시스템 및 접근통제 정책을 적용한 시스템들로 구분하여 성능 및 안전성을 측정하였다.(단, 실험을 위해 리눅스의 기본 접근통제 정책인 DAC정책을 우회하도록 커널을 수정하였다.)

2. lmbench 성능 분석

lmbench 도구에 의한 성능 실험 결과는 표 5와 표 6과 같다. 표 5의 시스템 콜 실험 결과를 보면 비교대상인 베이스 리눅스 보다 정책이 적용되지 않은 제안된 보안구조의 성능은 실험 항목에 따라 0.14%~36.09%의 성능 저하가 발생하며, SELinux 경우는 제안된 보안구조와 비슷한 수준인 1%~33%의 성능저하가 발생한다 [5]. 가장 큰 성능저하를 보인 PIPE의 지연율의 경우 실제 성능의 차이는 1.7881로 시스템의 전체성능을 크게 저하시킬 정도는 아니다. 그러나 제안된 보안구조에 정책을 적용한 경우는 1.39%~58.88%의 성능저하를 보인다. 성능 저하의 대부분은 리눅스 커널 모듈로 작성된 정책메커니즘의 처리 모듈에서 발생한다. /bin/sh 프로세스를 실행하였을 경우의 성능을 비교하면 정책을 적용하지 않은 경우는 1.93%의 성능 저하를 보이며, 여기에 MAC, RBAC, 및 MAC+RBAC 정책을 적용한 경우는 각각 21.41%, 11.08%, 22.05%의 성능 저하를 보인다. MAC 정책만 적용한 경우가 RBAC 정책만을 적용한 경우보다 성능저하가 크지만 이는 실험을 위해 구현된 리눅스 모듈 프로그램의 MAC구현 메커니즘이 RBAC구현 메커니즘보다 복잡하게 구현되었다는 것을 의미하지만, 일반적으로 MAC메커니즘이 RBAC메커니즘보다 성능이 좋지 않다는 것을 의미하지 않는다. 실험결과에 의해 정책을 적용하지 않는 제안된 구조의 시스템 콜의 성능저하는 평균 10%미만이며, 정책을 한 개만 적용할 경우 보다 정책을 많이 적용할 경우 성능저하가 더 크게 발생하는 것을 알 수 있다.

표 5. lmbench 시스템콜 실험 결과

단위(μs)

실험대상 / 테스트 항목	리눅스 커널 2.6.13	제안된 구조	제안된 구조 + MAC	제안된 구조 + RBAC	제안한 구조 +MAC +RBAC
simple syscall	0.1314	0.1321 (0.53%)	0.1336 (1.67%)	0.1371 (4.33%)	0.1434 (9.13%)
simple stat	1.7653	1.7684 (0.17%)	1.8588 (5.29%)	1.8700 (5.93%)	1.9893 (12.68%)
simple open/close	2.1328	2.3645 (10.86%)	2.6520 (24.34%)	2.5278 (18.52%)	2.9055 (36.22%)
Select on 10 fd's	0.6675	0.6684 (0.13%)	0.6768 (1.39%)	0.6695 (0.29%)	0.6942 (4.00%)
Select on 10 tcp fd's	0.7089	0.7206 (1.65%)	0.7366 (3.90%)	0.7208 (1.67%)	0.7418 (4.64%)
Signal handler installation	0.5184	0.5194 (0.19%)	0.5216 (0.61%)	0.5210 (0.50%)	0.5414 (4.43%)
Singal handler overhead	1.6052	1.6577 (3.27%)	1.6931 (5.47%)	1.6261 (1.30%)	1.7328 (7.94%)
Protection fault	0.6766	0.6844 (1.15%)	0.6865 (1.46%)	0.6852 (1.27%)	0.7166 (5.91%)
Pipe latency	4.9534	6.7415 (36.09%)	7.6765 (54.9%)	7.5030 (51.4%)	7.8703 (58.8%)
Process fork+exit	182.6452	198.6786 (8.77%)	218.2308 (19.48%)	202.4074 (10.81%)	220.6000 (20.78%)
Process fork+execve	586.2000	601.6000 (2.62%)	739.8571 (26.21%)	675.2500 (15.19%)	743.8571 (26.89%)
Process fork+/bin/sh -c	1761.2500	1795.3333 (1.93%)	2138.3333 (21.40%)	1956.3333 (11.07%)	2149.6667 (22.05%)
File /usr/tmp/XXX write bandwidth	49589KB/s	48300KB/s	49882KB/s	49257KB/s	49075KB/s

표 6은 lmbench.도구를 이용하여 64개의 프로세스들에 대한 컨텍스트 스위치 실험 결과이다. 괄호안의 수치값은 $T_{overhead}$=(실험대상들의 실험값-리눅스 커널 2.6.13의 실험값)/리눅스 커널 2.6.13의 실험값 * 100 의 계산값이다. 정책을 적용하지 않은 제안된 구조의 컨텍스트 스위치 성능저하는 약 2%미만이며, MAC 정책을 적용한 경우 3.9%~16.5%, RBAC 정책을 적용한 경우 4.17%~17.3%, MAC+RBAC 정책을 적용한 경우 4.66% ~21.4%로 나타났다. 또한, 데이터 세그먼

트 크기가 클수록 성능저하 비율이 감소함을 알 수 있다. 시스템 콜 실험결과와 마찬가지로 성능저하는 커널 모듈로 작성된 정책의 메커니즘들의 처리 시간과 비례하며, 적용되는 정책이 많을수록 성능저하의 비율도 늘어난다.

표 6. lmbench 컨텍스트 스위치 실험 결과 (프로세스의 개수 = 64)

단위(μs)

실험대상 \ Data Segment Size	0K	4K	8K	16K	32K	64K
리눅스 커널 2.6.13	4.10	7.00	9.01	12.41	18.21	30.44
제안된 구조	4.15(1.2)	7.01(0.1)	9.19(1.9)	12.44(0.2)	18.36(0.8)	30.45(0.03)
제안된 구조 + MAC	4.78(16.5)	7.45(6.4)	9.85(9.3)	12.90(3.9)	19.26(5.7)	31.65(3.97)
제안된 구조 + RBAC	4.81(17.3)	7.58(8.2)	9.88(9.6)	12.97(4.5)	19.43(6.6)	31.71(4.17)
제안된 구조 + MAC + RBAC	4.98(21.4)	7.64(9.1)	9.98(10.7)	13.21(6.4)	19.56(7.4)	31.86(4.66)

결론적으로, 제안된 구조는 다양한 접근통제정책을 리눅스 커널에 동적으로 적용할 수 있는 장점이 있으나 적용된 정책의 수가 많아지는 경우 성능저하가 발생하는 단점이 있다. 그리고 데이터 세그먼트 크기가 64K일 경우에서와 같이 정책을 2개 이상 적용한 경우의 오버헤드는 정책을 한 개씩만 적용한 경우의 오버헤드의 합보다 작다.

즉, $T_{overhead}(MAC+RBAC) < T_{overhead}(MAC)+T_{overhead}(RBAC) \Leftrightarrow 4.66 < 3.97+4.17$ 이다. 따라서 제안된 보안구조는 정책을 많이 적용할수록 성능저하가 발생하지만 전체적인 오버헤드는 개별 정책들의 오버헤드들의 총합보다 작다.

3. 안전성 분석

LTP 도구를 사용하여 각 시스템 콜에 대한 테스트와 기본 커널 모듈에 해당하는 파일 시스템, Direct IO, 메모리 관리, IPC 내구성, 스케줄러 내구성 등의 테스트를 한다. LTP 도구의 안전성 실험은 해당 시스템 콜에 비정상적인 인자를 전달하고 이에 대해 적절한 에러 코드와 반환값을 바탕으로 테스트 성공과 테스트 실패를 결정한다. 표 7은 LTP 도구를 사용하여 시스템 콜의 안전성을 측정한 결과이다.

해당 항목들의 값은 (실패한 테스트 항목의 개수 / 테스트 항목의 총 개수)를 의미한다. 그리고 테스트 항목은 리눅스 커널 버전 또는 패키지 버전에 따라 다르다.

표 7. LTP 실험 결과(FAIL/총개수)

테스트 항목	리눅스 커널 2.6.13	제안된 구조	제안된 구조 + MAC	제안된 구조 + RBAC	제안된 구조 +MAC +RBAC
시스템 콜	3/687	3/687	3/687	3/687	3/687
NPTL	0/1	0/1	0/1	0/1	0/1
파일시스템	0/55	0/55	0/55	0/55	0/55
Direct IO	0/28	0/28	0/28	0/28	0/28
메모리 관리	0/21	0/21	0/21	0/21	0/21
Pipe 내구성	0/8	0/8	0/8	0/8	0/8
스케줄러 내구성	0/3	0/3	0/3	0/3	0/3
pty 내구성	0/3	0/3	0/3	0/3	0/3
math library	0/10	0/10	0/10	0/10	0/10

LTP 도구를 사용한 안전성 분석 결과를 분석하면 시스템 콜의 안전성 테스트 687개중 3개의 테스트를 실패하였다. 테스트에 실패한 시스템 콜들은 각각 madivse02, ioperm02, fcntl23 이며, 시스템콜 다음의 숫자는 LTP 도구에서 수행하는 테스트 번호에 해당한다. fcntl23 시스템 콜 테스트의 예를 들면, LTP 도구에서 수행하는 fcntl 시스템 콜 테스트 중 23번째에 해당하는 테스트를 실패하였다는 의미이다. 따라서 제안한 보안구조는 리눅스 커널 2.6.13과 같이 동일한 시스템 콜들에서 LTP 테스트를 성공하지 못하였으므로 해당 시스템 콜들에 대한 보완이 필요하며, 그 외의 경우에는 비교 대상인 리눅스 커널 2.6.13과 동일하게 안전성을 보장한다. 또한, 메모리 관리 및 스케줄러 내구성 테스트 등에 해당하는 기본 커널모듈 테스트를 모두 통과하였으므로 제안한 보안구조는 커널 모듈들의 안전성을 보장한다.

Ⅵ. 결 론

본 논문에서는 가상접근통제 시스템을 이용하여 다양한 접근통제를 보안 운영체제에 적용할 수 있는 접근통제 보안 구조를 제안하였다. RSBAC의 보안구조는 SELinux의 Flask 보안 구조와 상당히 유사하지만, 이들 사이에는 몇몇 차이점이 존재한다. Flask 보안 구조가 시스템의 주체/객체에 할당되는 보안 레이블을 관리하는 책임을 접근통제 판정 기능을 담당하는 보안서버(Security Server)에서 수행하는 반면 RSBAC에서는 보안 레이블을 유지하기 위해 ACI(Access Control Information) 모듈을

사용한다. 이 때문에 RSBAC에서 보안 레이블을 표현하는 것은 Flask 보안 구조만큼 자유롭지 못하고 보안정책에 의존적이다. 그리고 RSBAC의 보안 구조는 접근제어 판정 결과를 캐싱하는 메커니즘을 제공하지 않으며, SELinux와 비교해 런타임 상태에서 접근통제 정책/메커니즘을 교체하는 것이 용이하지 않다. 또한, SELinux는 모든 접근통제 정책을 수용하기는 어렵고, 모델에 대한 충분한 이해가 없는 관리자가 정책을 사용하기 어렵게 되어있다. 그리고 RSBAC이나 SELinux 보안구조에서는 정책충돌의 개념이나 해결방법에 대한 고려가 되어 있지 않다.

따라서 본 논문에서는 다양한 접근통제모델을 운영체제에 동적으로 쉽게 적용할 수 있는 방법과 접근통제기능을 커널로부터 분리하는 방법, 그리고 복수의 접근통제 모델들 사이에 발생하는 정책충돌의 문제를 정의하고 해결하는 방법을 제시하였다. 이를 위해, 본 논문에서는 리눅스 보안운영체제를 위해 가상접근통제시스템 개념이 적용된 접근통제 보안구조를 제안하고 lmbench와 LTP 벤치마크프로그램들을 이용하여 성능과 안전성을 측정하여 그 결과를 분석하였다. 제안된 보안구조는 가상접근통제시스템을 추가함으로써 다양한 접근통제 모델들을 계층적인 트리구조로 통합하여 하나의 접근통제 모델처럼 보이게 하여 이질적인 접근통제 모델을 운영체제에 쉽게 적용할 수 있다. 또한, 접근통제 모델에서 사용되는 공통적인 접근통제 인터페이스를 제공함으로써 정책개발자들이 쉽게 정책 구현과 관리할 수 있는 환경을 제공한다. 또한, 공통의 접근통제 인터페이스를 통하여 동적으로 접근통제 모델을 구현하여 운영체제에 적용할 수 있다. 그러나 적용되는 접근통제 모델들이 많아질수록 성능저하가 증가하는 단점이 있다.

향후, 제안한 보안구조를 이용한 보안운영체제를 개발하고 이를 공개소스기반의 보안운영체제로 발전시켜 외국 제품이나 기술과 차별화되는 보안기술로 발전시켜야 한다. 그렇기 위해서는 제안된 보안구조의 단점인 적용되는 통제모델의 수가 많아짐으로써 발생하는 성능저하의 단점을 보완하는 연구가 필요하다.

참 고 문 헌

[1] 김정녀, 손승원, 이철훈, "안전한 운영체제 접근제어 정책에 대한 보안성 및 성능 시험", *정보처리학회논문지*, 제 10-D권 제 5호, Aug. 2003.

[2] 홍기융, 김재명, 홍기완, "Secure OS 보안정책 및 메커니즘", *정보보호학회지*, 제 15권 제 4호, Aug. 2003.

[3] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-Based Access Control," *ACM Transactions on Information and Systems Security*, Vol. 4, No. 3, pp. 224-274, Aug 2001.

[4] A. Ott, "The Rule Set Based Access Control (RSBAC) Linux Kernel Security Extension," *8th Int. Linux Kongress*, Enschede 2001.

[5] P. Loscocco and S. Smalley, "Integrating Flexible Support for Security Policies into the Linux Operating System," *In Proceedings of the FREENIX Track: 2001 USENIX Annual Tec. Conference*, June 2001.

[6] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, and J. Lepreau, "The Flask Security Architecture: System Support for Diverse Security Policies," *In Proceedings of the Eighth USENIX Security Symposium*, pp. 123- 139, Aug. 1999.

[7] A. Ott, "Rule Set Based Access Control as Proposed in the Generalized Framework for Access Control approach in Linux," *Master's thesis, University of Hamburg*, pp. 157, Nov. 1997.

[8] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," *IEEE Computer, Vol 29, No 2* , pp. 38-47, 1996.

[9] L. Mcvoy and C. Staelin, "lmbench: Portable Tools for Performance Analysis," *In Proceedings of USENIX Annual Technical Conference*, Jan. 1996.

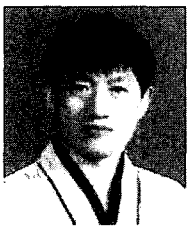
[10] M. D. Abrams, K. W. Eggers, L. J. L. Padula, and I. M. Olson, "A Generalized Framework for Access Control: An Informal Description," *In Proceedings of the Thirteenth National*

Computer Security Conference, pp. 135-143, Oct. 1990.
[11] M. D. Abrams, L. J. L. Padula, and I. M. Olson, "Building Generalized Access Control On UNIX," *In Proceedings of the 2nd USENIX Security Workshop*, pp. 65-70, Aug. 1990.
[12] C. P. Pfleeger and S. Lawrence Pfleeger, *Security in Computing*, PRENTICE HALL, 2002.
[13] D. Gollmann, *Computer Security*, John Wiley & SONS, 1999.
[14] S. Smalley, *Configuring the SELinux Policy*, Technical report, NSA, Feb. 2002.
[15] S. Smalley, C. Vance, and W. Salamon, Implementing SELinux as a Linux Security Module, Technical report, NAI Labs, May 2002.
[16] Medusa DS9, http://medusa.fornax.sk.
[17] The Linux Test Project http://ltp.sourceforge.net.
[18] The LMbench Project http://lmbench.sour ceforge.net.

〈著者紹介〉

김 정 순 (Jung-Sun Kim) 학생회원
1998년 2월: 전남대학교 전산학과 졸업
2000년 2월: 전남대학교 전산학과 석사
2000년 3월~현재: 전남대학교 전산학과 박사과정
〈관심분야〉 정보보호, 보안 운영체제, 컴퓨터 포렌식스

김 민 수 (Minsoo Kim) 정회원
1993년 2월: 전남대학교 전산통계학과 졸업
1995년 2월: 전남대학교 전산통계학과 석사
2000년 2월: 전남대학교 전산통계학과 박사
2000년~2001년: 한국정보보호진흥원 선임연구원
2001년~2004년: 전남대학교 연구교수
2005년~현재: 목포대학교 정보공학부 전임강사
〈관심분야〉 침입탐지, 컴퓨터 포렌식스, 보안 운영체제, 데이터마이닝 등



노 봉 남 (Bong-Nam Noh) 정회원
1978년 2월: 전남대학교 수학교육과 졸업
1982년 2월: 한국과학기술원 전산학과 석사
1994년 2월: 전북대학교 전산통계학과 박사
1983년~현재: 전남대학교 전자컴퓨터정보통신공학부 교수
〈관심분야〉 컴퓨터 네트워크 보안, 사이버 사회와 윤리