https://doi.org/10.5392/JKCA.2017.17.04.307

SELinux의 정책 재구성을 통한 성능 개선

Performance Improvements through Policy Reorganization in SELinux

고재용*, 최정인**, 조경연*, 이철훈*
충남대학교 컴퓨터공학과*, 한화시스템㈜**

Jae-Yong Ko(bbxiix@cnu.ac.kr)*, Jeong-In Choi(jeongin.choi@hanwha.com)**,
Kyung-Yeon Cho(c-ky10@cnu.ac.kr)*, Cheol-Hoon Lee(clee@cnu.ac.kr)*

요약

SELinux는 Linux의 대중화에 힘입어 사용자가 쉽게 접근할 수 있는 보안 운영체제로 알려져 있으며, 다양한 보안 운영체제 시스템의 참고자료나 임베디드, 서버와 같은 다양한 시스템에 적용되고 있다. 하지만 SELinux 커널 모듈 활성화에 따른 성능 오버헤드를 고려하지 않은 채 SELinux를 적용할 경우, 해당 시스템 전체의 성능저하로 이어질 수 있다. 본 논문에서는 SELinux 커널 내부에서 성능에 직접적으로 영향을 끼치는 요인에 대해 기술하였으며, SELinux 커널에 대한 변경 없이 정책을 재구성하는 것만으로도 성능을 개선시킬 수 있음을 보인다. 이는 보안 관리자나 개발자가 SELinux를 적용할 때 참고 자료로 사용될 수 있다.

■ 중심어 : | 보안운영체제 | SELinux | 정책 | 오버헤드 | 타입 강제 |

Abstract

SELinux is known as a secure operating system that is easily accessible to users due to the popularization of Linux, and is applied to various security operating system references deployed on systems such as embedded systems and servers. However, if SELinux is applied without considering the performance overhead of activating the SELinux kernel module, the performance of the entire system may be degraded. In this paper, we describe the factors directly affecting the performance inside the SELinux kernel and show that it is possible to improve performance by simply reorganizing the policy without changing the SELinux kernel. This can be used as a reference when security administrators or developers apply SELinux.

■ keyword : | Secure OS | SELinux | Policy | Overhead | Type Enforcement |

I. 서 론

시스템이 복잡해지고, 다양해질수록 이에 대한 보안 침해 공격 또한 증가하고 있는 추세이다. 특히 시스템이 철저하게 보호되어야 할수록 사용자의 데이터와 시스템 정보에 대한 접근 권한 관리가 중요해진다.

IT 침해 공격 중에서 대표적인 해킹 공격으로는 하이재킹 공격이나 APT 공격이 있는데, 이러한 공격들의 목표는 관리자 권한의 획득이다. 하지만, 사용자 레벨의 보안 프로세스만으로는 만약 공격자의 공격시도가 진행 중이거나 공격이 성공되었을 때에 대한 대비하는 침해 대응 과정이 부족하다. 이러한 침해 대응 과정은 시

* 본 연구는 2016년도 한화시스템(주) 연구과제로 수행되었습니다.
접수일자 : 2017년 01월 13일
수정일자 : 2017년 03월 22일
심사완료일 : 2017년 04월 06일
교신저자 : 이철훈, e-mail : clee@cnu.ac.kr

스템 레벨의 보안 방법인 보안 운영체제를 사용해서 특정 자원에 대한 강제적인 접근 제어를 할 수 있으며, 이를 통해 시스템 내 중요한 정보들을 보호할 수 있게 된다.

다양한 시스템이 존재하지만, 특정 시스템에 세부적인 접근제어를 적용하는 것은 해당 시스템에 대한 높은 이해가 필요하기 때문에 보안 운영체제를 적용하는 것은 쉬운 일이 아니다. 시스템마다 보호되어야 할 자원이 다르다는 것은 그만큼 다양한 보호 규칙들이 존재할 수 있다는 것을 의미하기 때문에, 이는 보안 운영체제가 차지하는 메모리의 크기가 커지는 문제나 이로 비롯된 시간 지연의 문제를 피할 수 없게 된다. 특히, 최적화가 중요한 임베디드 분야나 시간 결정성이 중요한 실시간성 시스템[1], 대량의 접근제어가 이뤄지는 대규모 서버 등은 보안 운영체제를 적용하기에 앞서 적용하기 전과 후의 성능에 대한 고려를 해야 한다.

본 논문에서는 다양한 시스템에 사용되고 있는 리눅스의 대표적인 보안 운영체제인 SELinux의 성능 상 시간 지연이 발생하는 요인을 알아보고, 이를 개선하기 위한 방안을 제시한다.

본 논문의 구성은 다음과 같다. 2장에서 SELinux의 성능과 관련한 기본 개념을 설명하고 3장에서는 SELinux의 성능과 연관된 연구들을 살펴본다. 4장에서는 SELinux상의 시간 지연에 직접적인 요인이 되는 부분을 분석해보고, 5장에서는 SELinux의 성능을 개선하기 위한 방안을 제시하며, 6장에서는 제시한 방안에 대한 성능 분석을 진행한다. 마지막으로 7장에서 결론을 맺는다.

II. SELinux

이 장에서는 리눅스의 가장 대표적인 보안 운영체제인 SELinux의 구조와 접근제어 기법과 관련된 개념을 살펴본다.

1. SELinux LSM 구조

SELinux는 Flask 보안 구조를 기반으로 구현되어 있으며, 이는 기존에 리눅스가 가지고 있는 리눅스 보안 모듈(LSM, Linux Security Module)과 상호작용하여 동작한다. Flask 보안 구조는 정책 적용의 유연성과 계층적인 보안성을 위해 강제적 접근 제어 기법을 구현한 것이다[2]. [그림 1]은 Flask 보안 구조가 반영된 SELinux의 커널 구조를 나타낸 것이다.

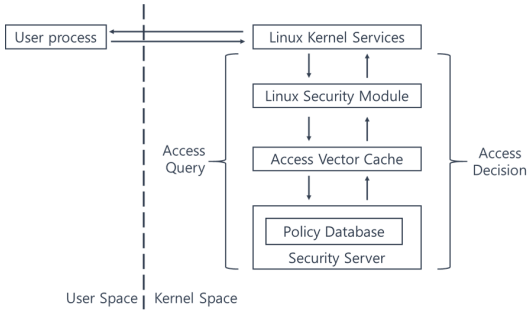

그림 1. SELinux Kernel Architecture

SELinux에 연관된 모든 시스템 명령들은 커널 영역으로 들어온 뒤 정책 서버에 대한 접근질의(Access Query)와 접근 결정(Access Decision)을 하게 된다. 이는 LSM에 존재하는 후킹모듈(Hooking Module)을 통해 시스템 콜을 가로채 접근의 허용 여부를 결정하는 과정을 거치게 된다. 이러한 과정은 SELinux가 비 활성화된 시스템과 비교했을 때, 시스템 콜 처리과정이 확장된 것이기 때문에 추가적인 시간 지연으로 인한 오버헤드가 발생할 수밖에 없다.

이러한 오버헤드를 줄이기 위해서 SELinux는 자체적으로 제공하는 접근 벡터 캐시(AVC, Access Vector Cache)를 사용하며, AVC를 통해서는 대략 7%의 성능 저하가 발생한다고 알려져 있다[3].

하지만 이는 AVC에 대한 최적의 상황만을 고려한 오버헤드의 수치이며, 캐시에 미스(Miss)가 발생한 상황에서는 접근 질의를 결정하기 위해서 정책 데이터베이스(Policy Database)에 접근하기 때문에 최악의 경우에는 전체 시스템의 성능 악화로 이어질 수 있다[4]. 접근에 대한 질의는 먼저 AVC내에 캐시된 규칙들과 비교하여 접근이 허용되었는지를 결정한다. 만약 AVC에서 캐시 미스가 발생할 경우, 접근 질의는 정책 서버 내부의 정책 데이터베이스에 존재하는 규칙들과 비교를 하게 된다. 이때, 추가적인 시간 지연이 발생하며, 이렇게 AVC의 캐시 미스를 통해 발생된 시간 지연의 크기

는 각 시스템 콜에 해당하는 단위 시간당 수행 시간의 지연 크기에 영향을 끼친다. 그뿐만 아니라 N 번의 횟수로 누적된 오버헤드의 크기는 전체 시스템의 수행 시간을 증대시켜 시스템 성능 악화의 원인이 된다.

2. SELinux의 강제적 접근 제어(Mandatory Access Control)기법

SELinux의 강제적 접근 제어는 주체(Subject)에 의해 객체(Object)에 대한 접근 질의가 발생했을 때, 정책 서버에 존재하는 접근 제어 정책과 필수적으로 비교 하게 된다. 그 후, 그에 따른 접근 결정을 리눅스 보안 모듈을 통해 주고받는 과정을 거친다.

여기서 주체란 특정 접근 질의를 하게 되는 사용자나 프로세스를 의미하며, 객체는 주체가 접근할 수 있는 모든 자원을 의미한다. 객체는 시스템 내에서 자원의 속성에 따라 클래스가 구분되어 정의되어 있으며, 주체는 보안 컨텍스트(SC, Security Context)라고 불리는 특정 데이터를 통해 자신만의 보안 정보를 갖게 된다[5]. 이 보안 컨텍스트에는 SELinux의 강제적 접근 제어 기법과 연관된 정보가 담겨져 있다.

SELinux의 강제적 접근제어는 타입 강제방식(TE, Type Enforcement), 역할기반 접근제어 방식(RBAC, Role-based Access Control), 다중 등급 보안(MLS, Multi-Level Security), 다중 범주 보안(MCS, Multi-Category Security)이 있다. 그중에서도 SELinux의 대표적인 접근제어 방식인 TE는 SELinux에 의해 영향을 받게 되는 주체와 객체가 미리 지정된 타입과 정책에 규정된 규칙에 의해 제어되는 것을 의미한다.

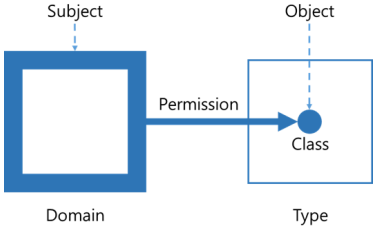

그림 2. AV Rule with Type Enforement

[그림 2]는 SELinux 내에서 TE와 연관된 정책 내 규칙을 의미하는 AV(Access Vector) 규칙을 나타낸 것이다. 주체가 가진 타입은 도메인(Domain)이라고 부르며, 해당 도메인을 가진 프로세스는 특정 타입(Type)을 가진 클래스(Class)의 객체(Object)에 접근하기 위한 권한(Permission)이 허용(Allow)되어야 한다는 것을 표현한 것이 AV 규칙이다[5]. 주체와 객체는 각각 타입을 가지고 있으며, 자신이 속한 타입의 AV 규칙을 따르면서 접근제어가 수행된다.

SELinux는 “All deny, allow some”이라는 원칙을 정책에 반영한다. 이러한 정책 내부에는 다양한 시스템 내부자원과 서비스에 대해 허용 방식이나, 타입, 클래스, 권한 등이 다양하게 구성되어 있으며 이들을 조합하여 수많은 규칙이 생성될 수 있다. 이렇게 생성된 규칙들은 정책 파일로 만들어져 SELinux 초기화 과정에서 보안 서버에 존재하는 정책 데이터베이스에 SELinux 파일 시스템을 통해 바이너리 형식으로 로드된다[5].

III. 관련 연구

SELinux 상에서의 오버헤드는 필수 불가결한 요소이기에 SELinux가 제공되어온 이래로 오버헤드를 줄이기 위한 연구가 진행됐다. 특히 SELinux를 특정 시스템에 적용하거나, SELinux에 새로운 기능을 추가하려고 할 때, 이를 비롯한 성능 측정 과정에서 이러한 오버헤드를 고려하지 않을 수 없다.

SELinux에서 자체적으로 제공하는 AVC 또한 이러한 연구의 일부분이며 다양한 관점으로 SELinux의 적용 후의 성능을 개선하려는 연구 또한 진행되고 있다. 일반적으로 SELinux의 성능과 관련한 연구는 SELinux와 같은 보안 운영체제의 적용 후 성능을 분석하거나 SELinux의 자체의 성능을 개선하는 연구로 구분될 수 있다.

1. 보안 운영체제 적용에 따른 성능 측정

국내에서도 보안 운영체제의 성능에 대한 연구가 진

행되어 왔다. 그 중에서 김정녀(2003)의 안전한 운영체제에서 접근제어 모델에 따른 보안성과 성능 시험을 한 연구결과는 접근 모델 중 MAC를 적용한 모델이 일반적인 DAC보다 보안성이 우수하며 성능 면에서는 RBAC를 적용했을 때보다 수행 시간 능력이 7%정도 더 우수하다는 것을 알렸다[6]. 또한 고영웅(2005)의 보안 운영체제에 대해 오버헤드를 분석한 연구는 당시 상용 보안 운영체제 제품별 발생한 오버헤드를 측정하고 성능측정 방법에 대해 제시하였다[7]. 이러한 연구들의 공통점은 다양한 접근제어 모델로 구성된 보안 운영체제를 특정 시스템에 적용하고 그로 인해 비롯된 오버헤드의 크기를 분석하는 데 의의가 있다.

2. SELinux가 사용된 시스템의 성능 측정

리눅스에 포함된 SELinux는 제조사에서 제공된 기본적인 정책을 단순히 활성화하는 것만으로도 다양한 시스템 내에서 강제적 접근제어를 통해 시스템에 영향을 미친다. 시스템에서 효과적으로 접근제어를 실시하기 위해 다양한 시스템에서 SELinux를 활용하는 방법에 대한 연구가 존재한다.

임베디드 장치에서 SELinux를 적용한 Nakamura(2008)의 연구결과와 리눅스 기반 임베디드 장치에서 SELinux를 적용한 Vogel(2008)의 연구는 정책의 크기를 줄이는 것을 통해서 임베디드 장치와 같은 제한적인 환경에서 SELinux를 적용하였다[8][9]. 이를 위해서는 커널 내 메모리 자원을 최적화하거나, 해당 장비에서 사용하는 시스템 자원과 구조를 정확히 파악하고 필요한 규칙만으로 구성된 정책을 사용함으로써 정책을 크기를 줄이는 방법이 적용된다. 해당 연구들의 공통점은 한정된 자원을 접근제어하기 위해 정책 크기를 축소했던 방법들이 성능이 개선되는 효과가 있었다는 것이다.

SELinux는 네트워크를 사용하는 서버와 연관되어 적용될 수 있다. 서버에 SELinux를 적용시키기 위한 연구는 다양한 접근 방법으로 진행됐다. Yokoyama(2009)의 인터넷 서버에 보안 운영체제를 적용시키기 위해 정책의 크기를 줄이는 방법에 대한 연구는 네트워크 취약점에 해당하는 규칙을 생성하고 전체 정책의 크기를 줄인 연구이다[10]. 또한 Rao(2009)는 다양한 이해관계자들이 존재하는 분산 환경에서 서로 다른 정책을 동적으로 적용하는 보안 프레임워크를 연구함으로써, 오버헤드의 크기가 다른 정책들을 동적으로 적용하는 방법을 제안하였다[11].

성능상의 관점으로 바라보았을 때, 해당 연구들의 공통점은 SELinux를 적용하기 위해서 정책의 크기를 변경하였던 부분이 성능 측면에서 시간 지연이 개선하는 효과를 가져왔다는 특징을 가지고 있다.

3. SELinux의 자체 성능 개선과 관련된 연구

SELinux는 AVC를 사용함으로써, 정책 서버에 접근을 질의하고 결정하는 과정을 단축시켰다. 그러나 보호해야 할 자원의 종류와 수가 많고, 다양한 주체가 존재하는 시스템에 SELinux를 적용할 경우 정책은 거대해지고 복잡해질 수 있다. 또한 임베디드 환경에서는 성능과 메모리의 최적화가 중요한 이슈이기 때문에 정책의 크기와 복잡성이 가져오는 오버헤드 관리가 중요하다.

SELinux의 자체적인 성능을 개선하기 위한 여러 연구가 진행되어 왔다. A.Kalyanasundaram(2012)의 멀티코어 프로세서 환경에서의 SELinux의 데이터 병렬처리에 대한 활용 연구는 AVC 캐시 미스일 때, 보안 정책내의 규칙의 수가 성능에 결정적으로 영향을 미친다는 것을 확인하였으며, 정책 서버내의 병목현상을 해소하기 위해 멀티코어 프로세서를 활용하여 성능을 개선하였다[4]. Fiorin(2012)의 임베디드 환경에서의 SELinux 사용을 위해 하드웨어 가속과 관련한 구현을 연구한 결과는 임베디드 환경에서 SELinux의 접근질의를 하드웨어 가속화로 성능을 향상시키고 소비전력에서도 향상하였다[12]. 이러한 연구들은 SELinux 커널 모듈에 새로운 기능을 추가함으로써, 성능을 개선하였다.

IV. SELinux의 시간 지연 요인 분석

관련 연구를 통해서 SELinux의 성능은 정책의 복잡성으로 비롯된 크기와 밀접한 연관이 있다는 것을 알

수 있다. 국내의 기존 연구들은 보안운영체제라는 솔루션을 특정시스템에 적용하거나, 접근제어모델에 따른 성능 측정을 수행하였다. 그 이외의 연구는 SELinux를 사용함에 있어서 특정 시스템 환경의 한정된 메모리 자원과 소비전력 문제를 해결하기 위해 혹은 적용되는 정책의 복잡성을 해소하기 위해서 규칙의 수를 줄이는 방안들이 제시되었다. 이러한 연구들은 정책 전체의 크기를 줄임으로써 발생되는 성능 개선의 효과를 간접적으로 누린 것이며, 직접적으로 성능에 대해 영향을 끼치는 부분을 고려한 것은 아니다. 이에 비해 A.Kalyanasundaram(2012)의 멀티코어 프로세서 환경에서의 SELinux의 데이터 병렬처리에 대한 활용 연구나 Fiorin(2012)의 임베디드 환경에서의 SELinux 사용을 위해 하드웨어 가속과 관련한 구현을 연구한 결과는 커널을 수정하여 실제 접근질의과정에 대한 성능 개선을 수행하였지만, 리눅스 커널을 재구성해야하거나 성능 향상을 위한 장치를 추가 하는 방법이 요구된다[4][12].

정책의 크기를 축소하는 것이나 리눅스 커널을 수정하는 것은 시스템에 대한 세밀한 분석이 요구되며, 일반 사용자나 개발자가 시스템 전체를 분석하기 위해서는 많은 시간과 비용이 소모된다. 또한 SELinux의 장점인 세밀한(fine-grained) 접근제어와 권한 분할에 있어서 정책의 크기는 정책의 복잡성과 보안성에 연관이 깊어 세심한 관심을 기울여야 한다[13]. 그뿐만 아니라 시스템 자체의 성능만을 위해 추가적인 장비를 도입함으로써 발생할 수 있는 금전적인 비용 또한 고려할 사항이 된다.

본 연구는 SELinux의 정책상의 요인과 구조상 접근질의 과정 분석을 통해 시간 지연이 발생하는 직접적인 요인을 분석한다. 그 후, 기존에 수행되었던 연구와는 다르게 커널 자체를 변경하거나 정책크기를 줄이는 등의 추가적인 행위 없이 본 연구에서 제시하는 정책 내 구성요소를 재구성하는 방법만으로도 성능 개선을 이끌어 낼 수 있음을 보인다.

1. SELinux의 정책상의 요인

SELinux상의 주체와 객체는 각각 주어진 타입을 통해 정책 내에 포함된 특정 규칙과 연관을 맺게 된다. 규칙은 특정 타입(Source_type)에 속한 주체가 주체의 타입과 같거나 다른 타입(Target_type)을 가진 객체의 클래스(Class)에 대하여 특정 권한(Permission)에 대하여 접근이 허용(Allow)된다는 것을 의미한다.

```
Allow source_type target_type : class permission;
```

그림 3. AV Rule format

SELinux의 정책은 주체와 객체가 타입에 의해 레이블링(Labeling) 되며 타입과 클래스, 권한에 따라서 다양한 규칙이 존재할 수 있다는 것을 의미한다. 시스템이 복잡하고 다양한 프로세스와 자원이 존재할 때, 그만큼 정책이 복잡해지고 거대해질 수 있다.

[표 1]과 [표 2] 그리고 [표 3]에 나온 수치들은 각각 Fedora사의 Minimum policy file[14], MLS policy file[14], Targeted policy file[14]의 3.13.1-102.el7_3.7버전과 3.13.1-60.el7_2.9버전을 분석하였고, TresysTechnology사의 Reference policy 2.20161023 version[15]을 분석하였다.

표 1. 각 정책 별 주요 규칙 수(Fedora사 정책 3.13.1-60.el7_2.9 version)

policy / rules	Fedora			Tresys Technology
	targeted	minimum	mls	reference
AV+TE Rules	121614+	66564+	84939+	128861+
RBAC Rules	442	241	160	239
MLS Rules	5666	1824	2719	40

표 2. 각 정책 별 주요 규칙 수(Fedora사 정책 3.13.1-102.el7_3.7 version)

policy / rules	Fedora			Tresys Technology
	targeted	minimum	mls	reference
AV+TE Rules	127285+	23980+	87417+	128861+
RBAC Rules	455	57	168	239
MLS Rules	5768	159	2952	40

SELinux의 정책 내에 존재하는 규칙은 크게 3가지로 나뉠 수 있는데. AV+TE 규칙, RBAC 규칙, MLS 규칙으로 나뉜다. RBAC 규칙은 정책 내에서 역할 전환(Role Transition)규칙을 의미하며, MLS 규칙은 범위 전환(Range Transition)규칙을 의미한다. 정책 내 모든 규칙들의 구성 비율을 백분율로 환산했을 경우, Targeted policy에서는 95%의 비율을 차지하며, Minimum policy에서는 99%의 비율을 차지한다. 또한 MLS policy와 Reference Policy에서는 각각 97%, 99%의 비율을 차지하는데, 이러한 비율은 해당 정책의 배포회사에서 정책 내 규칙을 구성하는 비율에 따라 달라질 수 있다. 예를 들어 Fedora사의 정책 구성요소 비율의 동향을 살펴보면 임베디드 장치와 같은 시스템에 적용을 위한 Minimum policy는 3.13.1-60.el7_2.9 버전보다 3.13.1-102.el7_3.7 버전이 AV+TE 규칙의 수가 대폭 감소한 것을 볼 수 있다. 이는 앞서 관련 연구에서 언급된 것처럼 임베디드 장치와 같은 시스템에서는 성능 상의 이유로 정책 크기를 줄이는 방법을 사용하게 되는데, 규칙의 수가 시스템 성능에 직접적인 영향을 준다는 것을 의미하며 실제 배포되는 기본 정책들 또한 그것에 맞게 정책이 수정되어 배포되고 있다는 것을 보여준다.

이처럼 AV+TE 규칙은 정책 내에서 가장 큰 비율을 차지하고 있는 규칙이다. AV+TE 규칙 내에서도 규칙은 사용 방식에 따라 여러 가지가 존재하며, [표 3]은 사용 방식에 따라 관련 규칙들의 수를 보여준다. 해당 규칙들의 수는 SELinux 정책 분석도구인 apol[16]을 활용하여 주요 규칙에 대하여 추출하여 분석한 최소한의 수치이며, 실제 적용된 정책 내 규칙의 수는 [표 1]과 [표 2]에 나타난 수치보다 더 많이 존재할 수 있다.

[표 3]은 AV+TE 규칙에서 Allow 규칙 수의 비율이 가장 큰 비율을 차지하고 있다는 것을 보여주며, SELinux의 성능에 영향을 주는 정책의 크기는 결국 정책 내에 Allow 규칙의 수와 가장 밀접한 연관이 있다는 것을 알 수 있다. 이러한 정책 내에 사용되는 모든 규칙은 시스템 부팅 시, SELinux를 초기화하는 과정에서 정책 데이터베이스에 로드된다.

표 3. AV+TE 규칙 내 각 규칙들의 수(Fedora사 정책 3.13.1-102.el7_3.7 version)

policy / AV+TE rule	Fedora			Tresys Technology
	targeted	minimum	mls	reference
allows	101234	20350	72632	103725
auditallows	157	41	8	25
dontaudits	8030	1934	6082	16881
neverallows	0	0	0	0
type_transitions	17755	1621	8605	8153
type_members	74	21	53	61
type_changes	35	13	37	16
총합	127285+	23980+	87417+	128861+

2. SELinux의 구조상의 요인

SELinux는 [그림 1]처럼 LSM 구조를 사용하여 시스템 콜을 후킹 하는 방식으로 접근 질의가 이뤄지면서 시간 지연이 발생한다. 시간 지연의 크기는 SELinux의 구조상 AVC를 사용하여 접근하는지, 정책 데이터베이스에 접근하는지에 따라서 각각 다르게 발생하게 된다.

AVC를 통한 시간 지연은 SELinux를 사용하는 데 있어 발생 가능한 최소한의 시간 지연으로, SELinux를 사용하는 최선의 성능을 의미한다. 반면에 캐시 미스가 발생하여 정책 데이터베이스에 접근하는 경우에는 시간 지연이 더 크게 발생할 수 있는데, 이때 정책의 크기가 시간 지연의 크기를 좌우한다. 시스템이 대규모화되거나 세부적인 보안을 위해 마련된 정책들이 생성되는 과정에서 정책의 크기가 커지게 되고 시간 지연이 커질 확률 또한 증가하게 되면서 시간 지연의 크기는 수십 배 이상 증가할 수 있다[4].

그 이유는 정책 데이터베이스에 정책이 로드되는 방식과 큰 연관이 있다. AVC나 정책 데이터베이스는 규칙을 효율적으로 관리하기 위하여 해시테이블과 연결리스트의 구조를 동시에 가지고 있는 체이닝 해시 테이블 구조를 사용한다.

[그림 4]와 같은 구조로 되어 있는 SELinux의 해시테이블은 각 노드가 규칙을 저장하고 있고, 각 노드을 비교하여 접근 질의를 처리한다. ε은 노드의 위치, N은 각 해시 엔트리에 연결된 연결리스트의 길이를 의미하고, $1 \leqq \varepsilon < N$일 때, 이러한 해시테이블은 규칙 비교시 O(N)의 시간 복잡도를 보인다. 이때, N의 크기는 규칙의 수

와 해시테이블의 엔트리 수에 직접적으로 영향을 받는다. 예를 들어 정책 크기가 고정적일 때, 체이닝 해시 테이블의 해시 엔트리의 개수가 각 슬롯의 연결 리스트의 크기나 메모리 사용효율에 반비례하다. 즉, 해시 엔트리의 개수가 적을 때에는, 연결리스트의 길이가 상대적으로 길어지기 때문에, 최악의 경우에 발생하는 시간 지연 또한, 그만큼 증가할 수 있다. 반면에 해시 엔트리의 개수가 많을 경우에는 엔트리의 개수에 맞게 전체적으로 분포되어 연결리스트의 길이가 짧아져 최악의 경우에 발생되는 지연 시간이 작아지지만, 메모리 효율면에서는 떨어질 수 있다. 그러므로 해시 엔트리의 개수를 조절하는 것은 시스템 요구사항을 고려하여 적용하여야 한다.

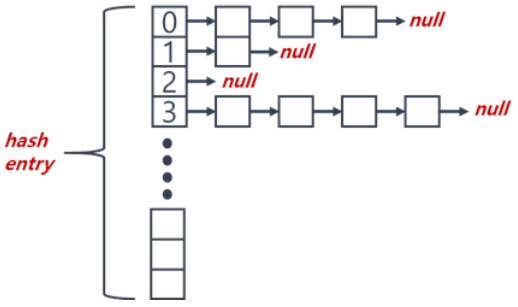

그림 4. Chaining Hash Table 구조

하지만 정책 사이즈를 줄이는 것만이 성능을 개선한다고 보장할 수 없다. 그 이유는 규칙과 관련된 정보가 삽입되는 구조와 연관이 깊기 때문이다. 정책 데이터베이스 내에서 규칙을 구성하는 정보는 접근 벡터 테이블(AVT, Access Vector Table)에 저장되며, 접근 벡터 테이블 노드(AVTN, Access Vector Table Node)의 형태로 삽입된다. [그림 5]는 AVT와 AVTN의 구조를 나타낸 것이다. AVTN은 키(Key)와 데이텀(Datum)으로 구성되어 있으며, Key값은 규칙내의 주체의 타입과 객체의 타입, 클래스 및 권한 정보, 허용 방식등과 같은 규칙 정보가 저장되는데, 이러한 정보는 체이닝 해시 테이블의 해시 인덱싱(Hash Indexing)과 노드 삽입(Node Inserting)에 사용된다.

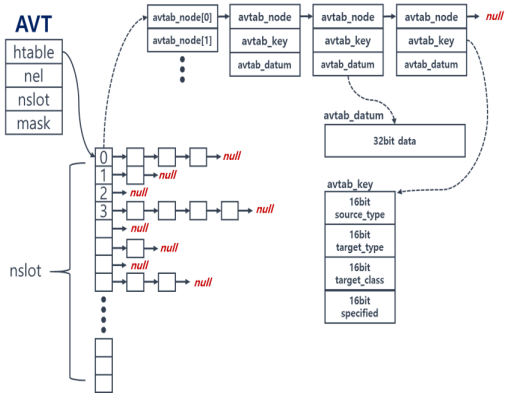

그림 5. Access Vector Table 구조

AVTN의 Key에 구성요소에 해당하는 Source_type, Target_type, Target_class는 16bit에 해당하는 값을 부여받은 후, 그 값을 기준으로 오름차순으로 정렬하여 삽입된다. 먼저 Source_type이 가장 작은 값을 가진 타입을 기준으로 오름차순으로 삽입되며, Source_type이 같을 경우에는 Target_type이 작은 순서대로 오름차순으로 삽입된다. Target_type이 같은 값을 가질 경우에는, Target_class 값이 오름차순으로 삽입된다. 결국, 규칙별 요소에 할당된 값이 작을수록 AVTN이 AVT의 앞쪽에 위치하게 된다.

이는 규칙이 정책 내에 없을 경우 해시 테이블의 연결리스트의 길이인 N개의 비교를 함으로써 발생하는 시간 지연을 줄이고자, 삽입 시 오름차순으로 정렬하여 삽입하게 되고, 특정 주체의 접근의 해당하는 Key 값이 같은 값이 아닌 자신보다 큰 값을 비교하게 될 경우 노트 탐색을 중지한다. 이러한 구조는 접근 허용(Access Allow) 결정이나, 접근 거부(Access Denied) 결정을 더욱 빠르게 처리할 수 있다.

AVT내에서 AVTN의 위치는 접근 질의에 대한 결정을 내리는데 있어 성능 상 중요한 의미를 가진다. AVTN의 위치가 타입에 의존적이라면, 규칙 수를 조절하는 것과 같이 정책 크기를 변화시키지 않더라도 특정 타입에 대하여 더 빠른 결정을 내림으로써 시간 지연을 줄일 수 있다는 것을 의미한다. 그 이유는 각 타입에 부여된 값이 작은 값일수록 AVT의 연결리스트의 앞쪽에 위치하게 되며, 해당 타입에 속하는 주체는 다른 규칙

에 비해 우선적인 타입 값 비교를 통해 접근 질의 결정을 내릴 수 있기 때문이다. 만약 시스템 성능 상 접근 질의가 잦거나 중요한 프로세스에 높은 우선순위를 가진 가장 작은 값을 부여한다면, AVC에서 캐시 미스가 발생하더라도, 정책의 크기에 상관없이 최대한 빠른 접근 질의 결정을 내릴 수 있게 된다.

V. SELinux의 성능 개선을 위한 연구

1. Checkpolicy

SELinux의 정책은 두 가지의 방식으로 생성될 수 있다. 한 가지는 각 정책을 (*.pp)파일 형태로 로드 가능한 형태로 생성하여, 사용자가 시스템이 운용 중에도 정책을 유동적으로 변화할 수 있게 생성하는 Loadable policy 방식이며 Checkmodule[18]이라는 프로그램을 통해서 빌드된다. 또 다른 한 가지는 Monolithic Policy 방식이며 모든 정책들이 policy.29와 같이 policy.(version number)의 형태를 띄고 있는 하나의 바이너리 정책 파일로 합쳐져 시스템 부팅 중에 정책 데이터베이스에 로드되는 방식을 의미한다. Monolithic policy방식은 Checkpolicy[17][18]라는 프로그램을 통해 생성된다.

Checkpolicy는 커널에 로드될 수 있는 이진 표현 방식으로 SELinux 보안 정책을 컴파일하는 SELinux 전용 정책 컴파일러이다[17]. 정책 파일을 생성하기 위해서는 먼저 정책에 관련한 모든 정책 소스 파일(*.te, *.if, *.fc)들을 취합하여 하나의 policy.conf라는 정책 설정 파일을 생성하게 되는데 해당 파일에는 정책에 사용되는 구성 요소들의 선언과 규칙들이 나열되어 있다[2]. Checkpolicy 프로그램은 이 policy.conf 파일을 파싱하여 관련 요소를 검사하고 이진 파일로 변환한다[2]. 변환된 파일은 SElinux 파일 시스템을 통해 커널에 로드된다. [그림 6]는 Checkpolicy를 사용하여 정책을 생성하는 과정을 나타낸 것이다[2].

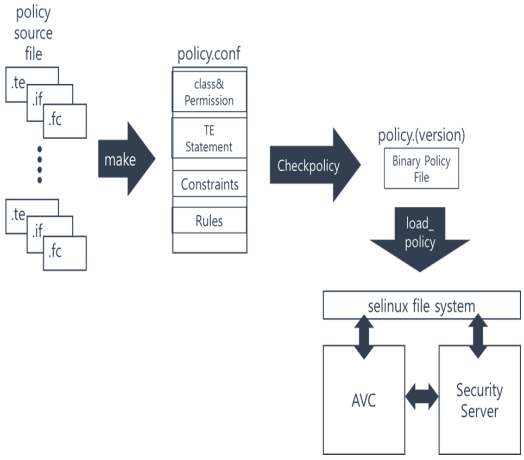

그림 6.Monolithic Policy방식을 통한 정책 생성과정

2. Libsepol

Libsepol은 SELinux 정책을 생성하기 위해 API를 제공하는 라이브러리를 말한다. 이것은 Checkpolicy나 Checkmodule, load_policy[18]와 같은 프로그램을 만드는 데 사용되며, 정책의 생성과 로드하는 과정에 관여하는 SELinux 라이브러리이다.

3. 성능 개선을 위한 정책 생성

기존에 존재하는 Libsepol 라이브러리나, 해당 라이브러리를 통해 생성된 Checkpolicy 프로그램은 개발자나 관리자가 AVTN의 위치를 결정할 수 없는 구조를 띠고 있었고, 타입의 역할과 그것이 시스템에 미치는 영향에 대한 고려가 전혀 없었다. 그렇기 때문에 실제 성능에 중요한 역할을 하게 될 타입의 주체가 특정 객체 클래스에 접근함에 있어 예측할 수 없는 시간 지연이 발생할 수밖에 없었다.

실제로 “bluetooth_conf_t”라는 타입은 “Targeted Policy” 상에서 각 타입에 할당 된 값 중 가장 낮은 값을 가진 타입으로서, AVT내에서 가장 선두에 위치하는 타입이다. 해당 타입을 가진 주체는 접근 질의 시 다른 타입에 비해서 AVC 캐시 미스가 발생해도 가장 빠르게 접근 질의를 처리할 수 있다. 하지만 다른 타입에 비해서, 접근질의 결정에 대한 횟수가 상대적으로 적어 성능 상 중요도가 낮은 타입이다.

정책이 커짐에 따라 AVC 캐시 미스가 발생했을 때, AVTN의 위치가 성능에 결정적인 요인으로 작용하게 되는 SELinux의 특성상 전체적인 시스템 성능과는 무관한 타입이 AVT내에서 높은 우선순위를 갖는 것은 문제가 될 수 있다. 그렇기 때문에 본 연구에서는 성능상 우선순위 기반의 타입 강제 방식을 제안하며 이는 “Priority-TE”라고 명명한다. 이는 타입에 대하여 성능상 우선순위를 부여하며, 중요도가 높은 타입에 한하여 작은 값을 부여함으로써, AVC 캐시 미스에도 보다 빠른 처리를 할 수 있게 한다.

특정 시스템 내에서 시스템 콜이 빈번히 발생하여 접근 질의를 자주하게 되는 중요한 프로세스들에 높은 우선순위의 타입 값을 부여한다면, 정책 내 타입을 재구성하는 것만으로도 시스템 성능을 개선할 수 있다.

본 연구에서는 정책 파일 내 구성요소에 의한 시간 지연 발생을 개선해보고자, Libsepol 라이브러리를 수정하였으며, 수정된 Libsepol 라이브러리를 통해 생성된 Checkpolicy 프로그램을 사용하여 기존의 방식과는 다르게 정책을 생성하였다. Libsepol 라이브러리에 존재하는 expand_module()함수는 policy.conf 파일에서 규정된 SELinux의 Type이나 Attribute, Alias와 같은 TE관련 구성요소들을 처리하는 함수이다. 여기서는 type_copy_callback()이라는 함수를 호출함으로써, Type에 해당하는 값을 직접 부여한다. 그러므로 type_copy_callback()함수는 AVT에서 AVTN의 위치를 결정하는 타입 값을 부여하는 핵심적인 부분이다. 해당 함수에서 타입에 값을 부여하는 부분을 수정하게 되면, policy.conf에 존재하는 타입들은 선언된 순서에 맞게 차례대로 값을 할당받게 된다.

수정된 Libsepol 라이브러리와 Checkpolicy 프로그램을 사용하게 되면 우선순위가 높은 타입들에 대하여 AVC 캐시 미스가 발생해도 상대적으로 빠른 접근 질의를 수행할 수 있는 정책이 생성된다. 이렇게 생성된 정책 파일은 시스템 부팅과정에서 커널에 정책이 로드될 때, 우선순위에 맞게 값이 부여되며 해당 타입들은 커널 내 정책 데이터베이스에 로드된다. 이를 통해 선언된 타입들이 로드된 후, 규칙들은 로드되는 과정에서 로드된 타입들의 값을 참조하여 커널 내 AVT에 AVTN으로서 삽입된다.

VI. Priority-TE를 적용한 정책에 대한 성능 분석 및 평가

이 장에서는 AVC 캐시 미스가 발생한 경우 낮은 우선순위와 높은 우선순위를 적용한 시간 지연에 대하여 비교 및 분석하고 평가를 진행한다.

1. 실험 환경

본 실험에서는 CentOS 7을 설치하고 리눅스 커널 3.10버전에 TresysTechnology 사에서 제공하는 참조 정책(Reference Policy)를 수정하여 사용하였다. Priority-TE 방식의 정책을 생성하기 위하여 Libsepol-2.6-rc2 라이브러리를 수정하였고, 수정된 라이브러리를 사용하여 Checkpolicy-2.6-rc2의 소스코드를 컴파일하여 타입에 우선순위를 부여하는 정책을 생성하는 프로그램을 제작하였다. AVT내의 AVTN의 위치를 변경하고, 실험의 용이성을 위하여 SELinux Monolithic policy 방식을 사용하여 정책을 생성하였다. 또한 정책 크기에 따른 성능 측정을 확인하기 위하여 AVT내에 규칙들을 추가하였으며, 동일한 실험 환경에서 같은 타입에 대하여 우선순위가 가장 높게 혹은 가장 낮게 부여되었을 때에 대한 성능 측정을 수행하였다. 정상적으로 생성된 정책 파일은 시스템 부팅 시, 리눅스 커널에 로드되었다.

본 실험은 사용자 레벨의 프로그램에서 특정 시스템 콜을 호출하였을 때, 시간 지연을 확인하기 위하여 여러 시스템 콜 중 create(), open(), write(), close(), remove() 시스템 콜을 각각 호출하였다. 한 프로시저가 다섯 개의 시스템 콜을 차례로 호출한 것을 1회라고 했을 때, 50,000회를 수행하였으며, 다섯 가지의 시스템 콜들이 각각 처리되는데 소요되는 시간을 측정하여 각 데이터 파일에 시간을 기록하는 프로그램을 작성하였다. 해당 프로그램은 리눅스의 /home/root/test/ 디렉터리에 생성하였으며, SELinux에 의해 부여된 타입은 “home_root_t” 타입이다. 이는 객체의 타입이

"home_root_t"라는 것을 의미한다. Root 권한을 가진 사용자는 기본적으로 Reference policy 정책 하에서 "staff_t"의 타입을 가진다. Reference policy 정책에는 "staff_t"를 가진 프로세스는 "home_root_t"타입을 가진 프로그램에 대하여 접근 권한이 없으므로 실행관련 권한을 추가한 규칙을 추가하여 정책을 생성 및 적용하였다.

본 실험은 성능 실험을 위해 "staff_t", "home_root_t"에 대하여 가장 높은 우선순위를 부여했을 경우와 가장 낮은 순위를 부여했을 경우로 나누어 실험을 진행하였다. 본 실험의 목적이 정책이 크기에 상관없이 성능 개선할 수 있음을 보이는 것이기 때문에, 실험의 효과를 극대화하기 위해서 AVT의 모든 엔트리에 대한 연결리스트의 길이를 증가하였다. 이를 위해서 1,000,000여건의 규칙을 추가하였으며 해당 규칙들은 AVT의 Masking 방식에 따라서 AVT의 모든 엔트리에 나뉘어 삽입되었다.

2. SELinux 정책 데이터 베이스 성능 실험

본 실험은 테스트 프로그램을 실행하는 주체의 타입인 "staff_t" 타입과 테스트 프로그램의 타입인 "home_root_t" 타입에 대하여 우선순위를 최상위로 주었을 경우와 최하위로 주었을 때, 우선순위를 제외한 모든 실험환경이 동일한 조건에서 진행되었다.

[그림 7]과 [그림 8], [그림 9]와 [표 4]는 테스트 프로그램을 실행하였을 경우의 각 시스템 콜 호출의 수행시간의 평균값을 나타낸 그래프와 표이다. [그림 7]과 [그림 8], [그림 9]에서 우선순위를 가장 높게 부여했을 경우와 우선순위를 낮게 부여했을 경우의 평균 처리 수행시간을 증감 비율로 환산했을 때, Create()는 우선순위가 높을 때는 우선순위가 낮을 때에 비하여 36%정도 성능 개선이 되었으며, Open(), Write(), Close(), Remove()는 각각 42%, 34%, 19%, 39%의 성능 개선율을 보였다.

AVT의 삽입 방식의 구조 상, 성능 상, 이점을 확보하기 위해 가장 고려되어야 할 것은 주요 프로세스에 대한 타입인 주체의 타입이다. 중요한 주체로 구분되는 사용자나 프로세스의 타입은 우선순위가 가장 높게 배치되어야하며, 객체의 타입의 우선순위보다 높아야 한다. 또한 접근되는 객체의 타입에 대하여, 성능상 중요한 파일들은 우선순위가 높은 객체 타입을 부여받음으로써, 정책이 커지는 상황에서도 일관된 성능을 보여줄 수 있다. 이는 우선순위가 높은 타입으로 구성된 규칙들이 AVT내의 연결리스트 앞쪽에 무리(Cluster)를 지어 구성된다는 것을 의미한다.

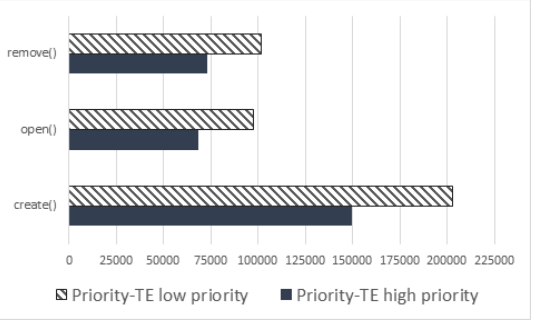

그림 7. remove(), open(), create()에 대한 시간 지연 비교

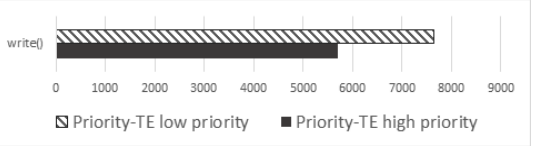

그림 8. write()에 대한 시간 지연 비교

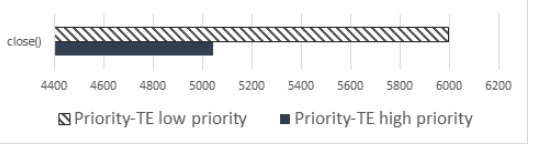

그림 9. close()에 대한 시간 지연 비교

표 4. 각 시스템 콜의 우선순위별 평균 수행 시간

policy / system call	Highest Priority type	Lowest Priority type
create()	149.481μ sec	202.763μ sec
open()	68.720μ sec	97.663μ sec
write()	5.703μ sec	7.658μ sec
close()	5.041μ sec	5.996μ sec
remove()	73.335μ sec	102.003μ sec

VII. 결론

SELinux는 보안을 위해 수많은 자원과 프로세스를 타입 라벨링한 뒤, 접근 제어를 수행한다. 세밀한 권한 분할과 규칙은 보안성을 극대화하기 위한 요소이지만 보안성을 위해서는 정책의 크기 증가 현상을 막을 수 없다. 리눅스의 기본적으로 탑재된 프로그램만을 위한 규칙의 개수는 수천여 개에 불과했지만, 현재 10만여 건으로 증가했다. 앞으로도 사용자의 활용 방법과 시스템의 특성 및 특징, 자원의 종류와 보안을 위한 TE의 활용방식을 적용했을 때, 수많은 규칙이 추가될 것으로 예상된다. 또한 사용자의 편의를 위해 SELinux의 규칙을 자동적으로 생성하는 도구들[19]은 규칙의 수가 증대되는데 이바지할 것으로 예상된다.

하지만 정책이 커져도 자주 사용되는 규칙들은 존재하기 마련이며, 이를 위해 AVC를 이용하여 보다 빠른 접근 질의 결정을 수행할 수 있도록 돕는다. 하지만 AVC의 크기는 메모리 관점에서 한계가 존재하며, AVC에서 캐시가 미스된 경우에는 최악의 경우 수많은 접근 질의 비교 연산으로 인한 시간 지연을 막을 수 없다. 특히 보다 빠른 처리가 중요한 프로세스의 접근이 캐시 미스 될 경우 문제가 발생할 수 있다. 시간 결정성이 중요한 경성 실시간 시스템과 같은 시스템에서는 시간 지연 문제로 인해 실시간 스케줄링이 실패할 경우 데드라인을 넘어 심각한 피해를 초래할 수 있기 때문에, 시스템 레벨의 보안을 적용할 경우 성능 상 시간 지연 문제를 고려해야 한다.

본 논문에서 제시하는 방법은 정책을 재구성하는 것으로도 성능에 영향을 끼칠 수 있다는 것을 알리는 연구이며, 크기가 작은 정책에서는 영향력이 미미할 수 있지만, 본 논문에서 제시한 방법을 적용하게 되면 정책의 크기가 커지더라도 우선순위가 높게 부여된 중요 프로세스의 타입에 한하여 일관된 성능을 보장할 수 있다.

그뿐만 아니라 TE방식에서 타입이 성능상의 우선순위를 가지기 위한 의미도 내포될 수 있음을 알리게 되는 것이며, TE를 적용하는 보안 운영체제들을 “Priority-TE” 방식을 통해 성능 개선을 할 수 있다. “Priority-TE”방식은 타입에 우선순위를 부여하여 성능상의 이점을 확보하기 위한 방식으로써, 본 연구에서 처음 제안되었다. 이러한 방식은 SELinux 가 적용된 리눅스 시스템뿐만 아니라, SEAndroid가 적용된 시스템에서도 적용될 수 있으며, 가장 큰 장점은 배포되는 상용 리눅스와 Android의 커널 변경 없이, 정책을 재구성하는 것만으로도 성능 개선을 이끌어 낼 수 있다는 것이다.

향후 연구과제로는 실시간 스케줄링이 요구되는 시스템에서 보안 커널이 활성화되었을 경우 주기성 확보를 위한 연구나, SEAndroid내에서 “Priority-TE” 방식이 적용된 정책 적용과 같은 연구가 필요하다.

참 고 문 헌

[1] 김주만, 송창인, 이철훈, “RTiK-Linux: 리눅스용 실시간 이식 커널의 설계,” 한국콘텐츠학회논문지, 제11권, 제9호, pp.45-53, 2011.

[2] Frank Mayer, Karl Macmillan, and David Caplan, *SELinux by Example*, 2006.

[3] http://www.crypt.gen.nz/selinux/faq.html#0.1

[4] A. Kalyanasundaram, B. B. Roy, and S. Rao, "Exploiting Data Parallelism in SELinux Using a Multicore Processor," in Proceedings of the 47th Annual National Convention of Computer Society of India (CSI), 2012.

[5] Haines, Richardm, *The selinux notebook*, 2014.

[6] 김정녀, 손승원, 이철훈, “안전한 운영체제 접근 제어 정책에 대한 보안성 및 성능시험,” 정보처리학회논문지, Vol.10, No.5, pp.773-780, 2003.

[7] 고영웅, “보안 운영체제의 오버헤드 분석,” 한국컴퓨터정보학회, Vol.10, No.2, pp.11-19, 2005.

[8] Yuichi Nakamura and Yoshiki Sameshima, "SELinux for consumer electronics devices," 2008 Linux Symposium, 2008.

[9] Björn Vogel and Bernd Steinke, "Using selinux security enforcement in linux-based embedded devices," Proceedings of the 1st international

conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[10] Toshihiro YOKOYAMA, Miyuki HANAOKA, Makoto SHIMAMURA, and Kenji KONO, Takahiro SHINAGAWA, "Reducing security policy size for internet servers in secure operating systems," IEICE transactions on information and systems, 2009.

[11] Vikhyath Rao and Trent Jaegerm, "Dynamic mandatory access control for multiple stakeholders," Proceedings of the 14th ACM symposium on Access control models and technologies, 2009.

[12] Leandro Fiorin, "Security enhanced linux on embedded systems: A hardware-accelerated implementation," 17th Asia and South Pacific Design Automation Conference, IEEE, 2012.

[13] 이재서, 김민수, 노봉남, "SELinux 보안 정책 복잡성 개선을 위한 보안 정책 설정 도구," 정보보호학회지, 제19권, 제2호, pp.43-52, 2009.

[14] https://fedoraproject.org/wiki/SELinux/Policies

[15] https://github.com/TresysTechnology/refpolicy

[16] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/4/html/SELinux_Guide/rhlcommon-section-0104.html

[17] https://fedoraproject.org/wiki/SELinux/checkpolicy

[18] https://github.com/SELinuxProject/selinux

[19] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security-Enhanced_Linux/S/Red_Hat_Enterprise_Linux/6/html/Security-Enhanced_Linux/

[20] 이상길, 이승율, 이철훈, "리눅스 사용자 영역에 실시간성 제공을 위한 미들웨어," 한국콘텐츠학회논문지, 제16권, 제5호, pp.217-228, 2016.

[21] 이상길, 이철훈, "멀티프로세서 기반 리눅스에 실시간성 지원 방안 연구," 한국콘텐츠학회 종합학술대회 논문집, pp.57-58, 2015.

[22] 정재엽, 박성종, 임재석, 이철훈, "임무컴퓨터를 위한 고가용 시스템의 구현 및 성능분석," 한국콘텐츠학회논문지, 제8권, 제8호, pp.47-56, 2008.

저 자 소 개

고 재 용(Jae-Yong Ko) 준회원

- 2016년 2월 : 충남대학교 컴퓨터공학과(공학사)
- 2016년 3월 ~ 현재 : 충남대학교 컴퓨터공학과 석사과정 재학

<관심분야> : 운영체제, 실시간 운영체제, 보안 운영체제

최 정 인(Jeong-In Choi) 정회원

- 1999년 2월 : 영남대학교 정보통신공학과(공학석사)
- 1999년 3월 ~ 현재 : 한화 시스템즈 TICN센터 수석연구원

<관심분야> : 운영체제, 미들웨어

조 경 연(Kyung-Yeon Cho) 준회원

- 2016년 2월 : 충남대학교 컴퓨터공학과(공학사)
- 2016년 3월 ~ 현재 : 충남대학교 컴퓨터공학과 석사과정 재학

<관심분야> : 운영체제, 실시간 운영체제, 보안 운영체제

이 철 훈(Cheol-Hoon Lee) 정회원

- 1983년 2월 : 서울대학교 전자공학과(공학사)
- 1988년 2월 : 한국과학기술원 전기 및 전자공학과(공학석사)
- 1992년 2월 : 한국과학기술원 전기 및 전자공학과(공학박사)
- 1983년 3월 ~ 1986년 2월 : 삼성전자 컴퓨터 사업부 연구원
- 1992년 3월 ~ 1994년 2월 : 삼성전자 컴퓨터 사업부 선임연구원
- 1994년 2월~1995년 2월 : Univ. of Michigan 객원 연구원
- 1995년 5월 ~ 현재 : 충남대학교 컴퓨터공학과 교수
- 2004년 2월 ~ 2005년 2월 : Univ. of Michigan 초빙 연구원

<관심분야> : 실시간시스템, 운영체제, 고장허용 컴퓨팅, 로봇 미들웨어