https://doi.org/10.5392/JKCA.2018.18.03.309

로드 가능한 모듈 정책을 사용하는 SELinux의 성능 향상을 위한 정책 재구성 방법

Policy Reorganization Method for Performance Improvements in SELinux using Loadable Module Policy

고재용, 이상길, 조경연, 이철훈
충남대학교 컴퓨터공학과

Jae-Yong Ko(bbxiix@cnu.ac.kr), Sanggil Lee(ask0137@cnu.ac.kr),
Kyung-Yeon Cho(c-ky10@cnu.ac.kr), Cheol-Hoon Lee(clee@cnu.ac.kr)

요약

SELinux는 리눅스를 사용하는 다양한 시스템에서 시스템 레벨의 보안을 위해 사용되고 있으며, 현재 IoT와 같은 기기 보안에도 활용되고 있다. 하지만 SELinux 적용에는 실행시간 저하에 대한 이슈가 내재되어 있기에 이를 해결하기 위한 다양한 연구가 진행되어 왔다. 본 논문에서는 SELinux를 활용하는 일반적인 방법인 Loadable module policy 방법이 적용된 환경에서, 정책 재구성을 통해 성능 개선할 수 있음을 보인다. 타입에 우선순위를 부여하는 Priority-TE 정책을 통해 접근질의테이블을 재구성함으로써 성능상 더 빠른 접근 질의가 필요한 타입에 대하여 보다 빠른 수행시간을 제공할 수 있게 된다. SELinux의 정책 적용 방법인 Monolithic 환경에서의 정책 구성 방법과의 차이점을 소개하고, 성능분석을 수행한다. 이는 보안 관리자나 개발자가 SELinux를 적용함에 있어 참고 자료로 사용될 수 있다.

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

Abstract

SELinux is used for system level security in various systems using Linux, and is now being used for device security such as IoT. However, since SELinux has inherent problems of execution time degradation, various studies have been conducted to solve this problem. In this paper, we show that performance can be improved through policy reconfiguration in the environment where the loadable module policy method, which is a general method using SELinux, is applied. By reconfiguring the access query table through the Priority-TE policy that gives priority to the type, it is possible to provide faster execution time for types requiring faster access query performance. This paper introduces the differences between SELinux policy configuration method in Monolithic environment and performance analysis. This can be used as a reference by security administrators or developers in applying SELinux.

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

* 이 연구는 충남대학교 학술연구비에 의해 지원되었음
접수일자 : 2018년 02월 07일
수정일자 : 2018년 02월 26일
심사완료일 : 2018년 02월 26일
교신저자 : 이철훈, e-mail : clee@cnu.ac.kr

I. 서 론

리눅스에서는 보안을 위해 LSM(Linux Security Module)을 사용한다. LSM[1]을 사용하는 보안 모듈에는 SELinux[2]를 포함한 SMACK[3], AppArmor[4]등이 있으며 이러한 보안 모듈들을 사용하여 접근제어와 같은 시스템 레벨의 보안을 수행한다. 대표적인 LSM인 SELinux는 가장 널리 사용되고 있는 리눅스 시스템 레벨 보안 방식이며 MAC(Mandatory Access Control)과 같은 강제적 접근제어기법을 적용하여 시스템 레벨에서의 강력한 보안 기능을 가지고 있다. SELinux의 MAC에는 RBAC(Role based Access Control), MLS(Multi Level Security)/MCS(Multi Category Security), TE(Type Enforcement)가 있다[5]. 리눅스 2.6버전 이후로 공식적으로 포함되어 리눅스를 사용하는 모든 장치나 시스템에서는 SELinux를 활성화하는 것만으로도 시스템 레벨 보안을 수행할 수 있다.

SELinux는 시스템 콜이 발생할 때, 해당 명령이 정책상으로 허용이 되어 있는지를 접근질의하고 이에 대해 판단하는 접근결정 과정이 추가되기에 실행시간 상의 오버헤드가 발생할 수밖에 없다. 이에 따라 SELinux의 실행시간 상의 오버헤드를 줄이기 위한 다양한 노력들이 있어왔다. TE 접근제어 기법에 사용되는 타입에 우선순위를 부여하고 성능상의 이점을 제공하는 Priority-TE(Priority-Type Enforcement)정책 생성 방식은 정책의 크기를 성능을 위해 일부러 감소시키거나, 커널의 수정이 없이도 성능을 개선시킬 수 있음을 보였다[6]. 하지만 기존의 연구는 Monolithic Policy 생성 방식을 따르는 SELinux에서만 가능한 방식이었으며, 사용자의 편의를 위해 제공되는 방식인 Loadable Module Policy 방식을 따르는 SELinux에서는 사용할 수 없다는 한계가 존재했다.

본 논문에서는 이러한 한계를 극복하기 위해 Loadable Module Policy 방식이 적용된 SELinux에서 Priority-TE 방식 적용을 위한 정책 생성 방법을 제시한다. 이는 IoT 장비나 군사, 의료와 같은 최적화가 중요한 리눅스 임베디드 분야[7]나 시간 결정성이 중요한 리눅스 실시간 시스템에 SELinux를 적용하는데 사용될 수 있다[8][9]. 또한 대량의 접근제어가 일어나는 대규모 서버 등은 SELinux를 적용하기에 앞서 성능에 대한 고려를 해야 하기에 참고 자료로 활용할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문의 연구 목적을 의미하는 Priority-TE에 대한 설명하고 3장에서 SELinux 정책 생성 방식을 분석한다. 4장에서는 일반적으로 제공되는 SELinux의 Loadable Module Policy 생성 방식에서 Prioity-TE방식을 제공하기 위한 알고리즘을 제시하며, 5,6장에서는 이에 대한 구현 및 테스트를 수행한 뒤, 7장에서 결론을 맺는다.

II. 관련 연구

SELinux는 본질적으로 모든 시스템 콜에 대하여 프로세스와 객체 파일에 대한 접근에 대한 규칙이 미리 정해져 있는지를 확인하는 접근 질의 과정을 거친다. [그림 1]과 같이 자주 접근되는 규칙에 대해 AVC(Access Vector Cache)를 사용하고 캐시 미스가 발생한 경우에는 정책 데이터베이스로의 접근 질의가 발생한다. 일반적으로, 리눅스에 탑재된 SELinux의 기본 규칙 수는 10만여 개 이상이며 그 수는 시스템에 따라 기하급수적으로 늘어 날 수 있다.

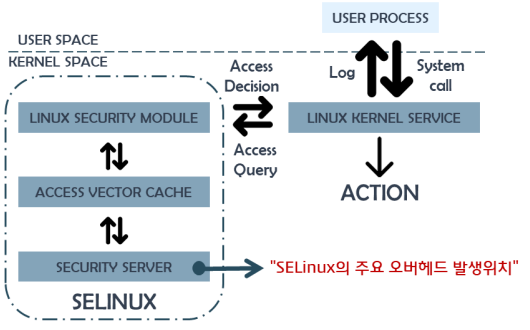

그림 1. SELinux의 접근 질의 구조 및 주요 오버헤드 발생 위치[10]

수많은 규칙들과 비교하기 위해서는 메모리 및 수행시간과 같은 제한된 요구사항이 존재한다. 기존의 SELinux 관련 성능 연구는 정책의 크기를 줄임으로써 접근질의 속도가 빨라졌다는 효과를 보인 연구[11][12]

나, 커널 자체를 수정하여 접근 질의를 빠르게 수행하는 연구들[13][14]이 진행되어왔다.

하지만, Fine-grained한 접근제어를 위한 SELinux의 본질적인 목표를 위해서는 정책의 크기를 단순히 줄이는 것은 그 목표와 부합하지 않으며, 커널 자체를 수정하는 것은 일반적으로 사용되어 배포되는 리눅스의 SELinux의 성능 개선에 적용시키기에는 일반 사용자의 부담이를 수밖에 없다.

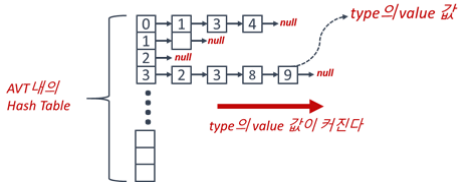

그림 2. SELinux Access Vector Table의 구조

이에 대한 대안으로 [6]에서 제시한 Priority-TE 방식이 있다. 해당 연구의 Priority-TE 정책 방식은 SELinux 타입에 우선순위를 부여하여 오버헤드가 발생하는 주요 지점인 정책 데이터베이스로의 접근질의 과정에서 빠른 접근 질의 및 결정을 수행하도록 하는 방식이다. 정책 데이터베이스 내에 규칙들은 [그림 2]와 같이 타입의 Value을 가지고 체이닝 해시 테이블에 노드로 존재하게 되며, 이때 타입의 Value가 작은 순서대로 정렬되어 삽입된다. 기존의 SELinux는 이러한 값에 의미를 부여하지 않아 성능이 중요하지 않은 타입이 더 빠른 접근 질의를 거칠 수 있다는 점이 있었고, Priority-TE 정책은 이 타입에 성능상의 의미를 부여하여 중요한 타입에 더 높은 우선순위를 부여함으로써 더 빠른 접근질의가 가능케 하였다. [6]에서는 이러한 성능 개선을 위해 Monolithic Policy 생성 프로그램인 Checkpolicy[15] 프로그램을 사용하여 Priority-TE 방식 정책을 구현하였다.

Priority-TE 방식의 장점은 정책의 크기가 커지더라도 성능이 중요한 프로세스에 한하여 타입의 우선순위를 높임으로써 접근질의를 빠르게 처리할 수 있다는 것이다. 또한 정책 바이너리 파일 내부의 요소들을 재구성하기 때문에, 커널을 수정하지 않고도 성능을 개선할 수 있다.

[6]의 연구는 이러한 Priority-TE 정책을 생성하고 그 성능개선 효과를 확인했다는 점에서 의의가 있다. 하지만 Monolithic Policy 방식으로 정책이 생성되는 과정은 SELinux 시스템에서 널리 사용되는 방식이 아니다. 그 이유는 시스템 운용 중에는 정책을 변경할 수 없어 갑작스런 접근 거부로 인해 시스템이 마비된다면 매번마다 SELinux 정책을 수정한 뒤 재부팅 과정을 거쳐야 한다는 단점이 있기 때문이다. 반면에 Loadable Module Policy 방식은 정책 모듈을 추가하고 삭제하는 과정을 통해 정책을 유동적으로 변경할 수 있다. 이러한 특징 때문에 시스템 운용 중에도 접근질의가 거부되거나 SELinux로 인해 정상적인 작동이 안 될 경우에, 정책 모듈을 수정하여 SELinux 정책을 시스템에 바로 반영하는 것이 가능하다. 이러한 방식은 시스템 관리자가 접근거부에 대하여 대응하는 부분에서 Loadable Module Policy 생성 방식이 Monolithic Policy 생성방식보다 더욱 유리하다는 장점이 있다. 특정 시스템에 맞게 유동적으로 정책을 수정하여 보안을 수행하기 때문에 일반적으로 Loadable Module Policy 생성 방식이 많이 사용된다. 또한 SELinux 정책을 수정하면서 그 접근 질의 현황을 시스템 운용 중에도 확인할 수 있다면, Priority-TE 정책이 적용될 경우 특정 타입의 우선순위 또한 시스템 운용 중에 변경하는 것이 가능해지며, 특정 프로세스의 타입의 우선순위를 높임으로써 AVC캐시미스 후에도 어느정도 일관된 수행시간을 보장할 수 있게 된다. 그렇기 때문에 Loadable Module Policy 정책 생성환경에서도 Priority-TE 정책 적용을 통해 성능 개선할 수 있는 방법이 요구된다.

III. 정책생성과정 분석

SELinux 정책은 SELinux 사용자 영역 프로그램에서 커널 영역과 같은 구조의 정책 데이터베이스를 만들고 이에 대한 정보를 정책 데이터베이스 바이너리 파일로 생성한다. 이 파일은 SELinux Security Server 내부의 커널 정책 데이터베이스에 바이너리 정보로 사용되기에, Priority-TE 기법이 적용된 정책을 적용하기 위해서는 먼저 사용자영역의 정책 데이터베이스에 타입에 우선순

위 값을 부여한 뒤 정책 데이터베이스 바이너리 파일을 생성하면 된다. 본 장에서는 Priority-TE 적용을 위해 SELinux 정책 생성과정을 분석한다.

1. Monolithic Policy 생성과정

Monolithic Policy 생성과정은 [그림 3]과 같이 정책 소스 파일로 불리는 여러 개의 .te(Type enforcement) 파일, .if(Interface) 파일, .fc(File context) 파일을 하나의 정책 설정파일인 policy.conf 파일로 생성한다.

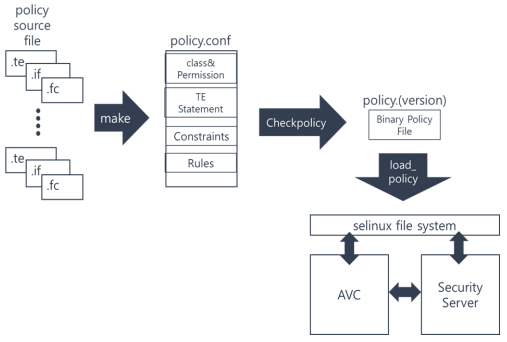

그림 3. SELinux Monolithic Policy 생성 구조

policy.conf파일은 생성되기에 앞서 하나의 정책 모듈을 이루는데 사용되는 .te, .if, .fc 파일들을 합하는 과정을 거치며 모든 정책 모듈들에 대하여 같은 과정을 수행하여 생성된다. 이렇게 생성된 policy.conf 파일은 하나의 거대한 텍스트로 구성된 설정파일로 사용되며 크게 선언 부(Statements)와 규칙 부(Rules)로 구성된다. 선언부에는 규칙에서 사용되는 유저(Users), 역할(Roles), 타입(Types), 클래스(Classes), 권한(Permissions), 제약사항(Constraints)등을 규정하는 내용이 포함되며, 규칙부에는 SELinux 규칙들이 포함된다.

policy.conf 파일은 Checkpolicy 프로그램을 사용하여 정책 바이너리 파일인 policy.version을 생성한다. 이 policy.version 파일은 Load_policy[16] 프로그램을 통해 커널 내부의 정책 데이터베이스에 로드된다.

Monolithic Policy 생성 환경에서는 정책 바이너리 파일을 생성할 때, 모든 정책 모듈로부터 합해진 policy.conf 파일만을 참고하여 생성하기 때문에 부분적으로 수정하더라도 전체 모듈들을 로드하고 합하여 하나의 policy.conf를 생성한뒤 정책 바이너리 파일(policy.version)을 생성한다. 또한 이렇게 만들어진 정책 바이너리 파일을 시스템 부팅 시에 로드하는 방식이기에 부팅 후에는 정책 정보 변경이 불가능하다. 시스템에 대한 완전한 분석과 정확한 정책을 생성 및 적용했다고 가정할 때 공격자가 정책을 임의로 수정할 수 없어 Loadable Module Policy 생성 방식보다는 더욱 안전한 방법으로 볼 수 있지만 시스템 관리자가 유용하게 사용하기에는 불편할 수 있다. 그 이유는 정책 소스 파일들이 각각의 바이너리 파일로 모듈화 되어 있지 않아 부팅 후에 임의대로 정책을 수정/삭제/추가 할 수 없기 때문이다. 시스템 관리자가 새로운 프로그램을 적용하거나 기존의 프로그램을 수정할 때 수정된 SELinux 보안 정책을 반영하기 위해 시스템을 매번마다 새로 부팅해야하는 번거로움이 있다.

2. Loadable Module Policy 생성 과정

Loadable Module Policy 생성 과정은 Monolitic Policy 생성 방식과는 다르게 각각의 정책 소스 파일들이 각각의 정책 모듈인 .pp(Policy Package) 파일을 생성한다. [그림 4]은 이러한 Loadable Module Policy 생성 방식을 나타낸 그림이다.

.pp 파일은 Checkmodule[17] 프로그램을 통해 .te 파일과 .if 파일을 합쳐 하나의 .mod(Module) 파일을 생성한 뒤 Semodule_package[18] 프로그램을 통해 File Context 정보를 담고 있는 .fc 파일을 .mod 파일과 합쳐 .pp 파일을 생성하게 된다. 기존의 Monolithic Policy 생성 방식과는 다르게 정책들이 모듈화 되어 있어 전체 정책 중 일부만 수정/삭제/추가하는 것이 가능하다. 이러한 방식은 부팅 후에도 유동적으로 정책을 수정할 수 있기에 SELinux의 보편적인 용도로 사용된다.

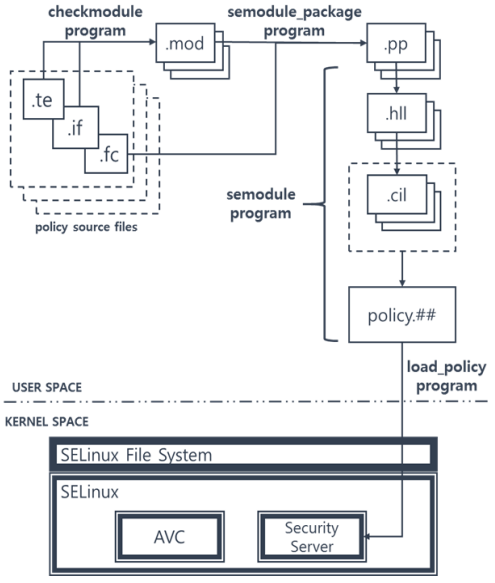

그림 4. SELinux Loadable Module Policy 정책 생성 구조

생성된 .pp 파일들은 각각의 목적에 맞게 구분되어 정책 파일로 존재한다. SELinux에서 기본적으로 제공되는 정책 모듈들은 407여개가 존재하며 kernel, admin, contrib, apps, roles, services, system 의 7가지 카테고리로 나뉘어 관리된다. 이 중 kernel 카테고리에 속하는 모듈들은 Monolithic policy 정책 생성 방식과 마찬가지로 base.conf 라고 불리는 설정파일로 하나로 합쳐져 관리된다. 이후 나머지 카테고리에 해당하는 모듈들이 base.conf 를 기준으로 나머지 정책 모듈 정보가 추가되는 식으로 정책 바이너리 파일이 생성된다.

.pp 파일들은 정책 바이너리 파일을 생성하기에 앞서, Semodule[19]이라는 프로그램을 사용하여 중간언어인 CIL(Common Interface Language)[20]로 번역되어 사용자 영역의 정책 데이터베이스를 생성한다. 이후 CIL 정책 데이터베이스 정책 수정이 완료될 경우 하나의 정책 바이너리 파일인 policy.version 으로 생성되어 Load_policy 프로그램에 의해 커널에 로드된다.

IV. Priority-TE 방식이 적용된 Loadable Module Policy 정책 생성 방법

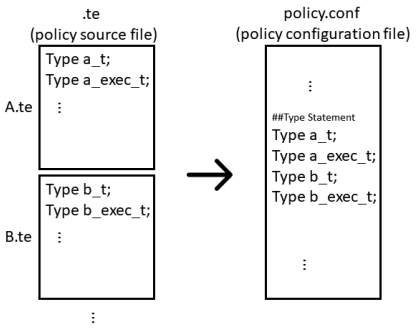

그림 5. SELinux 정책 소스 파일에서 Policy.conf 파일에 Type Statement가 기록되는 과정

Monolithic Policy 생성 방식과 Loadable Module Policy 생성 방식은 정책을 생성하는 구조가 달라 Priority-TE 방식이 적용된 정책을 생성하기에는 그 방법이 다를 수밖에 없다. Monolithic Policy 생성 방식에서는 [그림 3]에 나오는 policy.conf 파일 안에 각 .te파일들 내에 선언된 Type Enforcement Statements들이 [그림 5]과 같이 합쳐지며, 이후 선언된 순서대로 타입에 대한 값이 지정된다. 이 값은 규칙이 커널 내 정책 데이터베이스 내에 적재될 때, 작은 값을 가진 타입으로 구성된 규칙이 유리하기에 policy.conf에 선언되는 타입의 순서를 바꿔주는 것으로 타입의 우선순위를 조정 할 수 있다.

하지만 Loadable Module Policy 생성 방식은 타입이 선언되어 있는 소스 파일인 .te파일들이 policy.conf에 합쳐지지 않고 각각 정책 패키지 파일(.pp)로 바이너리화 된다. 이후 각각의 정책 바이너리파일들에서 정보를 읽어와 CIL 언어로 번역하여 CIL 정책 데이터베이스를 생성한다. CIL언어는 SELinux에서 사용하는 정책 컴파일러용 언어로 고수준의 언어로 작성된 정책 소스파일을 커널 내에서 사용할 수 있는 저수준의 언어로 번역하기 위해 사용되는 언어이다. CIL 컴파일러를 통해 컴파일된 모든 정책 설정 데이터들은 AST(Abstraction Syntax Tree)구조로 만들어진 뒤, CIL 정책 데이터베이스를 생성하여 관리 된다.

1. CIL 정책 데이터베이스 구조 분석

Loadable Module Policy 생성 방식에서는 Semodule 프로그램의 삽입 옵션(-i)을 사용하여 다수의 정책 패키

지 파일을 하나의 정책 바이너리 파일로 생성한다. 그 과정에서 먼저, 각 .pp 파일들은 Semodule 프로그램 내로 로드되고 그 안에 존재하는 Symbol들은 CIL Parse Tree Node로 만들어진다. 타입과 관련된 데이터는 정책 데이터베이스에서 사용되는 Key와 Value가 있으며 그중 타입에 해당하는 Key 데이터는 CIL Parse Tree에 존재하는 노드의 데이터를 CIL AST구조로 변환될 때 생성된다. 이 후 CIL 정책 데이터베이스를 빌드하는 과정을 거친다. CIL 정책 데이터베이스를 빌드하는 과정에서는 각 타입의 Key에 해당하는 Value를 부여받게 되는데 기존의 Semodule 프로그램에서는 타입 Value에 의미 없이 먼저 AST 노드 탐색된 순서에 따라 값이 부여된다. 하지만 Priority-TE 방식에서는 우선순위에 따라 타입 Value가 부여되어야 하며 SELinux 규칙들은 타입 Value가 작은 순서대로 정렬되어 커널 내 SELinux 정책 데이터베이스에 로드되어야 한다. 타입 Value는 [그림 2]와 같은 AVT 내에서 규칙 노드 (AVTN, Access Vector Table Node)의 위치에 영향을 주게 되며, 이후 AVC 캐시미스후의 주요 오버헤드를 감소시키는 역할을 하게 된다.

2. Loadable Module Policy 생성 과정에서의 Priority-TE 구현 방법 제시

본 연구에서는 Loadable Module Policy 생성 도구인 Semodule 프로그램을 수정하여 Priority-TE 정책 생성을 위한 방법을 제시한다.

Semodule 프로그램은 사용자가 입력한 옵션값에 대한 처리를 한 후, 플래그를 사용하여 정책의 변경이 있을 때, 정책을 재 컴파일 한다. 먼저 .pp 파일들을 로드한 후, CIL 컴파일러를 사용하여 사용자 레벨 정책 데이터베이스를 빌드 한다. 빌드 된 정책 데이터베이스를 정책 바이너리 파일로 생성하는 과정을 거친다. 이러한 과정에서 Priority-TE 정책 생성을 위해 모듈의 수정, 삭제 그리고 타입의 우선순위 변경에 대한 변경된 과정을 알고리즘으로 제시한다.

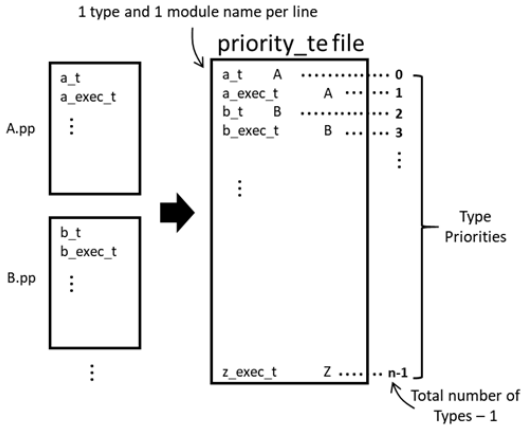

그림 6. priority_te 파일의 구조

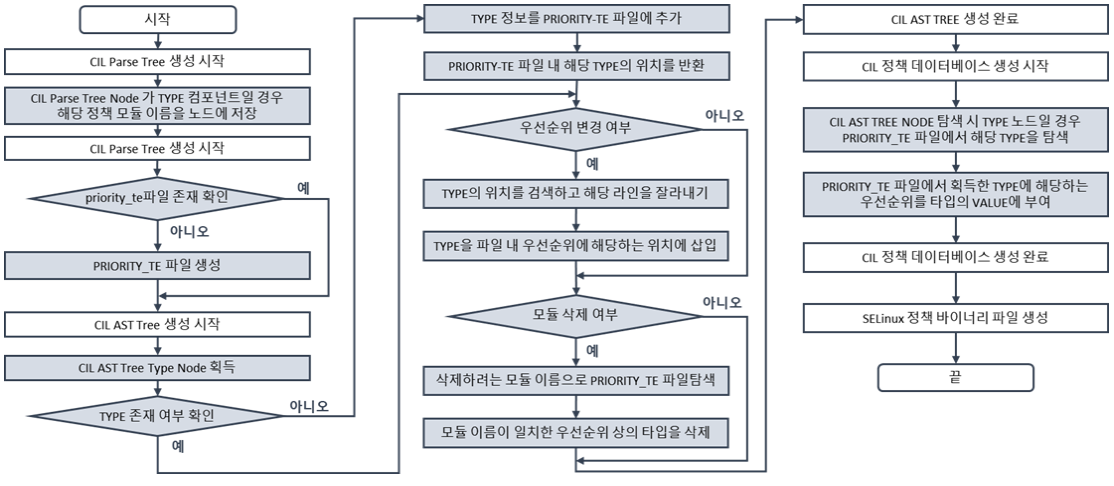

그림 7. Priority-TE 정책 생성을 위한 알고리즘

기본적으로 Priority-TE 정책 생성을 위해서는 파일 내에 우선순위의 순서대로 타입들을 기록하고 정렬하기 위한 우선순위 정보 파일인 priority_te 파일이 필요하다. [그림 6]은 이러한 priority_te파일을 나타낸 그림이다. 파일 내에서는 라인 당 하나의 타입과 해당 타입을 선언한 모듈의 이름이 기록되며 타입이 먼저 기록될수록 높은 우선순위를 부여한다.

별도의 파일을 작성하여 우선순위를 기록하는 목적은 모듈을 추가 및 수정, 삭제 그리고 타입의 우선순위를 변경하는 경우에도 기록을 통해 SELinux 정책의 변경에도 타입의 우선순위를 유지하고 타입들이 제대로 선언되어 있는지 검증하기 위함이다. [그림 7]은 Priority-TE를 제공하기 위한 알고리즘을 나타낸 것이다. [그림 7]에서 색상으로 칠해진 부분은 새롭게 추가된 알고리즘을 나타낸다.

Loadable Module Policy 생성 환경에서 Priority-TE 방식을 적용하기 위한 알고리즘을 추가할 경우, SELinux 관리자는 시스템 콜 수행시간 단축을 위해 주요한 프로세스가 가진 타입에 한하여 우선순위를 변경하여 성능을 개선할 수 있다. 뿐만 아니라, Loadable Module Policy를 사용하는 SELinux에서 정책을 추가 및 삭제하는 일반적인 과정에서 Priority-TE와 관련한 알고리즘만을 추가하였기 때문에 Priority-TE 방식을 사용하지 않는 SELinux와 마찬가지로 정책 수정을 할 수 있다.

2.1 모듈 추가 및 수정 시

모듈이 추가되는 경우는 정책소스 파일들을 컴파일 하는 경우나, 정책 모듈을 작성하여 모듈단위로 추가하면서 타입이 새롭게 추가되는 경우가 있다. 이 경우에는 Cil Parse Tree Node를 생성할 때, 현재 탐색된 Symbol 정보가 어떤 .pp 파일로부터 나왔는지 정보를 추가적으로 저장한다. 이후 CIL Parse Tree Node를 CIL AST Tree Node로 변환하는 과정에서 priority_te 파일을 생성하고 해당 노드가 타입 컴포넌트일 경우 priority_te 파일에 타입의 Key와 .pp 파일 이름을 기입한다. priority_te 파일이 이미 존재한다면 우선순위에 의한 타입 이름과 .pp 파일명만 기록한다. priority_te 파일 내의 데이터는 타입 노드가 탐색되어 priority_te 파일에 기록된 순서대로 한 줄당 하나의 타입 씩 기록된다. 라인의 위치는 곧 우선순위 값을 의미하게 된다. 기록되기 전 중복 기록되는 것을 방지하기 위하여 priority_te 파일 내에 타입이 기록되어 있는지 검증하는 과정을 거치고, 만약 기존에 없던 새로운 타입 이라면 priority_te 파일 내 제일 하단에 추가한다. 한번 기록된 타입은 모듈이 삭제되기 전까지는 유효하게 기록되어 사용된다.

2.2 타입 우선순위의 변경 시

모듈의 추가 없이 특정 타입의 우선순위를 변경하는 경우에는 CIL AST Tree가 생성되는 과정에서 타입에 Key값을 부여하는 부분에서 먼저 해당 타입이 priority_te 파일 내에 존재하는지 확인한 후, 있을 경우에 해당 타입과 관련한 정보를 요구된 우선순위 위치에 삽입하는 방식으로 priority_te 파일을 재구성한다. 이후. CIL 정책 데이터베이스가 빌드 되는 과정에서 Value을 부여 받을 때, 변경된 위치에 해당하는 우선순위 값을 부여 받으면서 타입의 우선순위가 재설정 된다.

2.3 정책 모듈 삭제 시

정책 모듈이 삭제되는 과정은 정책 모듈 내에 타입 Statement가 존재할 경우 해당 타입의 존재를 삭제하는 것을 의미한다. Semodule 프로그램의 정책 모듈 삭제 과정은 먼저 삭제하려는 정책 파일을 삭제한 뒤, 활성화된 정책 모듈 리스트에서 해당 정책 모듈을 배제하여 새롭게 CIL 정책 데이터베이스를 구성한다. 정책소스 파일 내에 선언된 타입과 규칙들은 같이 선언되고 관리되기에 타입만 삭제될 수 없도록 정책 모듈 삭제 시 타입도 삭제될 수 있도록 관리한다. Priority-TE 정책에서 우선순위를 priority_te파일을 통해 관리하고 있기에 삭제된 타입이 남아 있다면, 우선순위 상 잘못된 값을 부여할 수 있고, 타입의 총 개수가 일치하지 않아 타입 선언 검증 과정에서 문제가 발생할 수 있기에 priority_te 파일 내 에서도 삭제해야한다.

V. 구현 및 실험

Semodule 프로그램은 SELinux 정책 모듈들을 관리하

기 위해 사용된다. 이것은 Checkmodule 프로그램과 Semodule_package 프로그램에 의해 만들어진 .pp 파일을 삽입, 삭제, 또는 빌드 하는데 사용되고 정책 정보를 하나의 정책 데이터베이스용 바이너리 파일을 생성하는데 사용된다. Semodule 프로그램에서는 모듈 삽입 시에는 -i, 모듈 삭제 시에는 -r을 사용한다. 이와는 별개로 -priority 옵션이 존재하지만 그 목적이 달라 Priority-TE와의 기능을 구별하기 위해 -priorityTE 옵션을 추가하였다.

1. 모듈 추가 시

모듈을 추가하는 과정에서는 Libsemanage[21] 라이브러리의 semanage_direct_commit 함수를 수정하였다. 먼저 타입 우선순위를 기록하기 위한 priority_te파일을 생성하고, 기존에 priority_te파일이 있을 경우에는 이 과정을 생략한다. 이후 semanage_load_files 함수를 호출하면서 각 정책 관련 모듈들이 로드되고 cil_parser 함수가 호출되면서 CIL Parse Tree가 생성된다. 이때 cil_tree_node에 추가한 file_name 변수에 모듈 이름을 기록한다. 이후 Cil AST Tree가 만들어지는 과정에서 cil_gen_type 함수가 호출되는데, 이때 생성한 CIL 노드에 타입 정보를 추가하는 과정에서 priority_te 파일에 해당 타입이 있는지 검증한 뒤, 만약 없다면 타입 정보와 모듈 이름 정보를 추가한다. 이때 priority_te 파일내의 타입 위치가 곧 우선순위 값이 된다. 이후 cil_build_policydb 함수가 호출되어 정책 바이너리 파일을 생성하는 과정을 거친다. 이후 cil_type_to_policydb 함수가 호출되어 타입의 Value를 지정해준다. 앞서 기록된 priority_te 파일에서 타입 위치를 검색하여 타입의 존재 여부를 검색하고, 있을 경우 해당 위치를 정수 값으로 받아 타입 Value를 지정한다.

2. 타입 우선순위 임의변경 시

타입의 우선순위를 변경한다는 것은 타입 Value을 수정하는 일이기에 정책 데이터베이스를 재 빌드하는 과정을 반드시 거쳐야 한다. Semanage_handler를 통해서 do_rebuild 플래그를 활성화하고, Cil AST Tree가 생성되는 과정에서 변경되는 타입과 변경하려는 우선순위의 정보를 Semanage_handler를 통해서 넘겨받게 된다. 이후 cil_gen_type함수에서 -priorityTE 옵션에 의해 priority_te 플래그가 활성화되어 있다면 해당 타입을 priority_te 파일 내에 찾아서 변경하려는 우선순위의 위치에 삽입하게 된다.

3. 모듈 삭제 시

Semodule 프로그램 내에서 모듈을 삭제하는 행위는 먼저 삭제하려는 정책 모듈과 관련된 디렉토리 정보를 획득하고 관련된 파일을 삭제한 뒤, 새롭게 정책 바이너리 파일을 생성한다. 이때, priority_te파일 내에 선언된 타입을 삭제해야하기 때문에, priority_te파일 내에 기록된 정책 모듈의 path 정보와 비교하여 일치하는 타입들을 삭제한다.

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

이 장에서는 Loadable Module Policy 정책 생성 환경에서 Priority-TE 정책 생성과 이를 적용했을 경우 올바른 성능이 나오는지에 대하여 비교 및 분석을 진행한다.

1. 실험 환경

본 실험에서는 [표 1]과 같은 환경에서 SELinux Loadable policy 방식을 사용하여 실험 환경을 구축하였다. [6]의 연구와 마찬가지로 정책 크기에 따른 성능 측정을 확인하기 위하여 AVT(Access Vector Table)내에 규칙들을 추가하였다. 같은 타입에 대한 우선순위 변경에 따른 성능차이를 확인하기 위해, 최상위 및 최하위 우선순위 타입일 경우에 대한 성능 측정을 수행하였다. 테스트 프로그램이 특정 시스템 콜을 호출하였을 때, 시간 지연을 확인하기 위하여 create(), open(), write(), close(), remove() 시스템 콜을 각각 호출하였다. 각 시스템 콜이 호출되기 전과 후를 clock_gettime 함수를 사용하여 측정하였고 이 차이를 각각의 시스템 콜에 대한 데이터 파일에 기록하였다. 테스트 프로그램은 이 각각의 시스템 콜에 대하여 create(), open(), write(), close(),

remove()이 차례대로 수행되는 하나의 과정을 만들고 이를 50,000회 수행하여 실험을 진행하였다. 테스트 프로그램을 수행하는 주체의 타입인 “staff_t”와 “staff_t”가 수행하는 프로그램 객체의 타입인 “home_root_t”에 대한 규칙을 추가하였다. Loadable Module Policy 정책 환경에서도 Priority-TE 정책 적용에 대한 효과가 나오는 것을 확인하기 위해 1,000,000건의 규칙을 생성하여 추가하였다.

표 1. 실험 환경

실험 환경 조건	내용
운영체제	CentOS7, linux kernel-3.10
사용된 SELinux 정책	Reference Policy[22]
사용된 라이브러리	Libsepol-2.6-rc2[23], Libsemanage-2.6-rc2[21]
사용된 타입의 수	4438개
AVT내 규칙의 수	1,399,562개

2. SELinux 정책 데이터베이스 성능 실험

본 실험의 목적은 Loadable Module Policy 생성 방식에서도 Priority-TE 정책을 생성하고 이를 통해 성능 개선이 이뤄지는 것을 확인하는 것이다. 그렇기 때문에, Monolithic Policy 생성 방식에서 성능을 개선한 [6]의 연구와 같은 동등한 실험 환경에서 실험을 진행하고 이를 비교하여 올바르게 성능 개선이 이뤄졌는지를 확인한다.

표 2. 각 시스템 콜의 우선순위별 평균 수행 시간

시스템콜	High Priority	Low Priority	성능 개선 율
Create()	182.234 μ sec	270.654 μ sec	33%
Open()	75.803 μ sec	128.083 μ sec	41%
Write()	6.573 μ sec	15.151 μ sec	57%
Close()	5.733 μ sec	10.305 μ sec	44%
Remove()	81.629 μ sec	132.602 μ sec	38%

먼저 올바르게 Loadable Module Policy 정책 생성 방식으로 Priority-TE 정책을 생성했는지 여부를 확인한다. 이는 각각의 타입에 대하여 최상위 및 최하위 우선순위를 부여하고 동일한 환경에서 테스트 프로그램을 수행하였을 때 측정된 각 시스템콜의 평균 수행시간으로 그 차이를 비교한다. [표 2]와 [그림 8]은 테스트 프로그램을 실행하였을 경우 각각의 시스템 콜의 수행시간의 평균값을 나타낸 결과 자료이다. [표 2]과 [그림 8]을 통해 높은 우선순위를 가질 경우와 낮은 우선순위를 가질 경우에 대한 성능 차이를 볼 수 있으며, 높은 우선순위를 가질 경우에는 create(), open(), write(), close(), remove() 시스템 콜에 대하여 각각 33%, 41%, 57%, 44%, 38%의 평균 수행시간의 성능 개선을 확인 할 수 있다. 이는 많은 규칙이 있다고 하더라도 Priority-TE 정책을 사용할 경우 접근질의 결정시간이 향상됨을 나타낸다.

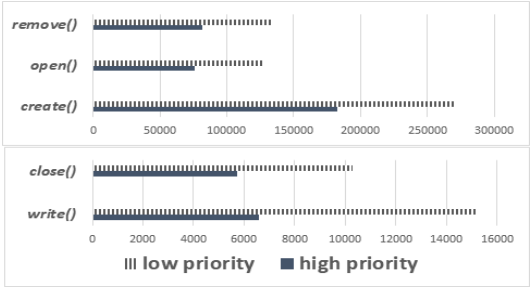

그림 8. 실험된 시스템 콜의 시간 지연 비교

다음으로는 Monolithic Policy 생성방식에서 생성된 Priority-TE 정책과의 비교를 수행한다[6]. [6]의 연구에서 나타난 성능 개선율은 36%, 42%, 34%, 19%, 39%로 본 연구와 비교했을 경우 유사한 성능 개선을 보인다. 특히 최상위 우선순위를 부여했을 경우 각 수행 시간은 각각 149.481μ sec, 68.720μ sec, 5.703μ sec, 5.041μ sec, 73.335μ sec로 [표 2]에서 나타난 수행시간과 유사하다[6]. 이는 AVT내 규칙 노드의 위치가 AVT의 연결리스트의 길이에 상관없이 가장 선두에 위치하기에 평균 수행시간에서의 유사점을 띈다. 반면에 write(), close()함수에 대해서는 본 연구에서 나타난 실험에서 최하위 우선순위를 부여했을 경우 더 긴 평균수행시간을 보인다. 그 이유는 [6]의 연구에서 사용된 Allow 규칙의 수와 본 연구에서 진행한 총 규칙의 수에 차이가 발생하여, AVT의 연결리스트의 길이가 달라 수행시간이 더 길어질 수 있다.

이러한 실험 결과는 Loadable Module Policy 생성과정에서도 성공적으로 Priority-TE 정책을 생성 및 적용함을 의미하며, 결과적으로는 SELinux의 Loadable Module Policy 생성 방식으로도 Priority-TE 정책을 생성할 수 있으며 이로 인해 성능을 개선할 수 있음을 나타낸다.

VII. 결론

본 연구에서는 SELinux의 성능 개선 문제와 Loadable Module Policy 생성 환경에서의 Priority-TE 방식의 정책 생성 문제를 해결하기 위해 Semodule 프로그램을 개선하였다. 본 연구에서 제안하는 방식은 기존 연구에서 사용한 Monolithic Policy 생성 방식과는 다르게 priority_te 파일을 통해 타입의 우선순위를 관리하는 방식을 도입하였다. 이로써, Loadable Module Policy 생성 환경에서도 간단하게 Priority-TE 정책을 생성할 수 있었으며 성능 실험을 통해 기존의 연구인 Monolithic Policy 정책 생성 환경과 마찬가지로 성능 개선을 확인할 수 있었다. 이는 보안 관리자가 성능상의 이유로 타입의 우선순위를 변경하여 Loadable Module Policy 생성 환경에서도 성능 개선을 할 수 있음을 의미한다.

향후 연구 과제로는 자주 접근 질의를 하는 프로세스의 타입을 관리하고 동적으로 타입의 우선순위를 변경하여 커널 내에서 Priority-TE 방식을 활용하여 SELinux의 오버헤드를 관리하는 연구가 있다.

참 고 문 헌

[1] https://en.wikipedia.org/wiki/Linux_Security_Modules

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

[3] https://en.wikipedia.org/wiki/Smack_(software)

[4] https://en.wikipedia.org/wiki/AppArmor

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

[6] 고재용, 최정인, 조경연, 이철훈, “SELinux의 정책 재구성을 통한 성능 개선,” 한국콘텐츠학회논문지, 제17권, 제4호, pp.307-319, 2017.

[7] 조경연, 고재용, 이상길, 이철훈, “임베디드 리눅스 시스템에 SELinux 적용 방법 연구,” 한국콘텐츠학회 종합학술대회 논문집, pp.371-372, 2017.

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

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

[10] 고재용, 조경연, 이상길, 이철훈, “로드 가능한 정책을 사용하는 SELinux의 정책 재구성을 통한 성능 향상,” 한국콘텐츠학회 종합학술대회 논문집, pp.359-360, 2017.

[11] 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.

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

[13] 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.

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

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

[16] https://fedoraproject.org/wiki/SELinux/load_policy
[17] https://fedoraproject.org/wiki/SELinux/checkmodule
[18] https://fedoraproject.org/wiki/SELinux/semodule_package
[19] https://fedoraproject.org/wiki/SELinux/semodule
[20] https://github.com/SELinuxProject/cil/wiki
[21] https://github.com/SELinuxProject/selinux/tree/master/libsemanage
[22] https://github.com/TresysTechnology/refpolicy
[23] https://github.com/SELinuxProject/selinux/tree/master/libsepol

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

이 상 길(Sang-Gil Lee) 정회원

- 2014년 2월 : 충남학교 컴퓨터공학과(공학사)
- 2016년 2월 : 충남학교 컴퓨터공학과(공학석사)
- 2016년 3월 ~ 현재 : 충남학교 컴퓨터공학과 박사과정 재학

<관심분야> : 실시간 운영체제, 임베디드 시스템

조 경 연(Kyung-Yeon Cho) 준회원

- 2016년 2월 : 충남대학교 컴퓨터공학과(공학사)
- 2018년 2월 : 충남대학교 컴퓨터공학과(공학석사)

<관심분야> : 운영체제, 실시간 시스템, 임베디드 시스템, 안드로이드 플랫폼

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