보안 운영체계를 위한 메커니즘 및 구조

이준원* 김현아* 심영철* 장인숙**
*홍익대학교 컴퓨터공학과
**국가보안기술연구소
{junlee, hakim, shim}@cs.hongik.ac.kr* {jis}@etri.re.kr**

Mechanism and Architecture for Secure Operating Systems

Jun-Won Lee* Hyun-A Kim* Young-Chul Shim* In-Sook Jang**
*Dept. of Computer Engineering, Honglk University
**National Security Research Institute

요 약

운영체계는 컴퓨팅 시스템에서 가장 중요한 보안 기능 제공자이다. 운영체계는 프로그래밍 환경을 제공하고, 멀티프로그래밍과 자원의 공유를 가능하게 하며 프로그램 행위에의 제한을 집행하는 기능을 수행한다. 이러한 운영체계에 신뢰성을 부여하기 위하여 1970년대부터 미국의 정부기관을 중심으로 하여 보안 운영체계에 대한 연구가 활발히 진행되어 왔다. 대부분의 연구는 강력하고도 유연한 접근 제어 정책을 운영체계 커널에 포함시키는 내용에 초점을 맞추어 왔다. 그러나 컴퓨팅 시스템이 인터넷 내에서 다른 컴퓨터들과 연결되어 사용되고 있고 내장형 시스템 내에서도 많이 사용되고 있는 등 그 사용 방법과 범위가 계속 변화하고 있으므로 운영체계의 보안에 대해서도 과거의 시야보다는 더 넓은 관점에서 다시 보아야 할 것이다. 본 논문에서는 보안 운영체계에서 제공하여야 할 보안 서비스 요구사항을 정의하고 다음 이들 요구사항을 제공하기 위해 필요한 보안 기능을 포함한 운영체계의 구조에 대한 설계방향을 제안한다.

1. 서 론

1970년대부터 미국의 정부기관을 중심으로 하여 보안 운영체계에 대한 연구가 현재까지 진행되어 왔다. 대부분의 연구는 강력하고도 유연한 접근 제어 정책을 운영체계 커널에 포함시키는 내용에 초점을 맞추었으나, 컴퓨팅 시스템이 인터넷 내에서 다른 컴퓨터들과 연결되어 사용되고 있고 내장형 시스템 내에서도 많이 사용되고 있는 등 그 사용 방법과 범위가 계속 변화하고 있으므로 운영체계의 보안에 대해서도 과거의 시야보다는 더 넓은 관점에서 다시 보아야 할 것이다.

보안 운영체계의 요구사항은 보안 정책(security policy)과 보안 모델(security model)로서 정의될 수 있고 기본 보안 메커니즘으로 사용자 식별과 인증, 강제적 접근제어(MAC), 임의적 접근제어(DAC), 객체의 재사용 보호, 감사 및 로깅 등이 요구되어진다. 그러나 운영체계는 프로그램의 크기가 매우 크기 때문에 보안 운영체계에 대한 개발이 체계적으로 이루어지지 못하는 것이 현실이다. 따라서 현재 상용으로 널리 사용되고 있는 운영체계들에서 새로운 보안 결함들이 계속 보고되어지고 있고 아무리 좋은 보안 도구들을 개발하여 이 운영체계 위에서 수행한다고 하여도 결국 많은 허점을 드러내고 있는 실정이다.

이에 본 논문에서는 보안 운영체계에서 제공하여야 할 보안 서비스 요구사항을 정의하고 다음 이들 요구사항을 제공하기 위해 필요한 보안 기능을 포함한 운영체계의 구조가 어떻게 정의되어야 하는지에 대한 설계방향을 제시한다.

2. 관련 연구

1970년대 초 미국 DARPA에서 Mach 시스템의 보안성을 강화시키기 위하여 TMach 프로젝트가 수행되었다. TMach 프로젝트는 Carnegie-Mellon 대학교의 연구용 운영체계인 Mach 시스템에 접근 제어 방식을 추가하여 신뢰성을 지닌 운영체계를 설계하는 것이었다[1]. 이 연구가 끝난 후 DARPA에서 NSA로 프로젝트가 이전되면서 NSA는 SCC사와 함께 TMach 프로젝트와 LOCK 프로젝트의 결과를 토대로 DTMach 프로젝트를 수행하였다. DTMach 프로젝트에서는 TMach에서는 볼 수 없었던 보안 서버를 추가하였다. 이 서버는 접근 제어에 대한 결정을 위한 독립적인 모듈로서 커널이 요청을 하면 자신의 정책에 따라 허가여부를 결정하게 되고 접근 제어 결정에 따라 커널은 집행을 담당한다. 이러한 정책 결정과 집행을 나누어서 관리함으로써 DTMach는 TMach보다 좀더 유연한 접근 제어를 가능하게 하였다. 또한 DTMach 커널은 정책 결정 캐쉬를 두어서 결정을 일정시간동안 저장하고 있어서 성능을 향상시켰다[2]. DTMach 프로젝트가 끝날 무렵 NSA와 SCC사는 DTOS 프로젝트를 통하여 DTMach 프로젝트를 계승하였다. 이 DTOS는 NSA가 주도한 보안 정책, 보안 메커니즘, 확장성 있는 보안 구조, 진보된 보안 시스템, 새로운 평가 방법론에 대한 유용한 프로토타입의 생성 등을 목적으로 하는 Synergy 프로젝트의 일부로서 DTOS 프로토타입의 구현으로 Synergy 프로젝트를 종료하였다[3]. NSA는 Synergy 프로젝트의 후속으로 SCC사, Utah 대학교와 공동으로 DTOS 프로젝트의 결과를 연구용 운영체계(Fluke)에 적용하여 구현하는 Flask 프로젝트를 수행하였다. Fluke에서는 이미 RVM(Recursive Virtual Machine)을 기반으로 메모리 접근 권한, 시스템 콜의 모니터링을 위한 마이크로커널 Capability 시스템 등의 보안 구조를 제공하고 있었는데 여기에 운영체계가 제공하는 모든 서비스들에 대한 접근 제어를 제공하는 것이 Flask이다. Flask는 보안 서버와 객체 관리자로 구성되어져 있는데, 보안 서버는 정책 데이터베이스나 관리자 인터페이스를 통하여 정책 결정을 하고 객체 관리자는 파일 관리자, 네트워크 관리자 등으로 나누어져 있으며 Flask는 이러한 객체 관리자를 위하여 보안 서버로부터 보안 검색 등에 대한 인터페이스, 보안 결정을 일시적으로 저장할 수 있는 AVC(Access Vector Cache)를 제공하며 보안 정책의 변경에 대한 통지를 받을 수 있도록 지원하고 있다[4]. 이 프로젝트를 수행한 후 NSA는 Linux에 Flask에 구현한 보안 모듈을 적용시키는 것을 목적으로 하는 SELinux 프로젝트를 현재까지 수행하고 있다. SELinux 프로젝트는 Flask의 보안 구조를 Linux 운영체계에 적용시킨 것으로 Flask와의 차이점은 보안 서버에서 TE와 RBAC(Role-Based Access Control)와 더불어 MLS를 제공하고 있고, Flask에서는 제공하지 않았던 사용자 식별 모델(User identity model)을 새롭게 제시하여 기존 Linux의 사용자 식별 속성을 변경 또는 제한을 두는 방법을 배제하고, 보안 문맥에 독립적인 사용자 식별 속성을 포함시켜 기존 Linux 접근 제어에 MAC을 완벽하게 접목시켰다[5].

위에서 언급했던 보안 운영체계에 관한 연구들에서는 운영체계의 신뢰성을 보장하기 위해 사용된 접근 제어 방식 이외에도 사용자 레벨에서 커널 레벨로 보내어지는 시스템 콜을 도중에 모니터링하고 있다가 시스템 콜의 허용여부를 판단하는 시스템 콜 모니터링 방법[6]이 있다. 이러한 시스템 콜 모니터링에는 두 가지 접근 방법이 있는데 첫 번째 방법은 사용자 레벨에서 시스템 콜 모니터링을 하는 것이고, 두 번째 방법은 커널 레벨에서 시스템 콜 모니터링을 하는 것이다. 이러한 방법들에는 각각 장단점이 존재하는데 사용자 레벨에서 시스템 콜 모니터링을 할 경우 좀 더 많은 유연성을 제공할 수 있다는 장점이 있지만 성능이 떨어진다는 문제점이 있다. 반면에 커널 레벨에서 시스템 콜 모니터링을 할 경우에는 속도가 빠르다는 장점이 있지만 이동성이 떨어지고, 운영체계가 복잡해진다는 문제점이 있다.

3. 보안 운영체계를 위한 보안 메커니즘

3.1 인증

인증은 사용자가 적절한 보안 속성과 연관되는 것을 보장하는 것으로 보안 속성에는 신원, 그룹, 역할, 보안 수준, 무결성 수준 등이 있다. 인가된 사용자의 명확한 식별과 사용자 및 주체와 보안 속성과의 연결은 의도한 보안 정책 수행에 매우 중요하다. 인증을 위해서는 인증실패, 사용자 속성 정의, 사용자 인증, 사용자-주체 연결 등의 요구사항들이 필요하다.

먼저, 인증 실패는 실패한 인증시도의 횟수와 인증시도 실패 시 취해야할 행동에 대한 것을 정의한다. 인증 실패 처리는 사용자 인증시도 실패 횟수가 명세된 값을 넘으면 세션설정 과정을 종료시키고, 세션설정 과정의 종료 후 관리자가 정의한 조건이 일어날 때까지 사용자의 계정이나 시도가 이루어진 진입점을 폐쇄 할 것을 요구한다. 또한 이러한 것을 모두 감사 기록할 수 있어야 한다. 두 번째로 사용자 속성 정의는 사용자 보안속성을 사용자와 연관시키기 위한 요구사항을 정의한다. 사용자 속성 정의는 각 사용자에 대한 사용자 보안속성을 개별적으로 관리될 수 있도록 하고 인가된 관리자는 사용자에 대한 추가적인 보안속성을 정의할 수 있다. 세 번째로 사용자 인증은 사용자 인증 메커니즘의 유형을 정의한다. 인증은 사용자가 허용될 모든 행동의 이전에 이루어져야 할 것을 요구하고, 인증 메커니즘은 위조되거나 복사된 인증 데이터의 사용을 탐지하고, 일회용 인증 메커니즘을 사용할 경우 재사용 방지를 위한 메커니즘을 요구한다. 다중 인증 메커니즘은 특정 사건에 대하여 사용자 신원을 인증하기 위해서 서로 다른 인증 메커니즘이 제공되고 사용될 것을 요구한다. 재인증은 사용자의 재인증이 필요한 사건을 명세하는 능력을 요구하고 인증 피드백 보호는 인증하는 동안 제한된 피드백 정보만이 사용자에게 제공될 것을 요구한다. 네 번째로 인증된 사용자는 일반적으로 주체를 활성화시키는데, 사용자의 보안속성은 이 주체와 연결되며 사용자-주체 연결은 사용자의 보안 속성과 사용자를 대신하여 행동하는 주체간의 연결을 생성하고 유지하기 위한 요구사항을 정의한다. 따라서 인가된 관리자는 주체의 디폴트 보안 속성을 정의할 수 있어야 하고 사용자 보안 속성과 주체의 성공한 연결과 실패한 연결에 대한 감사기록을 할 수 있어야 한다.

3.2 암호화

현재 암호화 알고리즘은 한정되어 있고 컴퓨팅 파워도 날이 갈수록 증가하고 있기 때문에 암호화가 풀릴 가능성이 많다. 그러므로 암호화에 있어서 암호화 메커니즘보다는 정책을 어떻게 결정할 것인지가 더욱 중요하다. 시스템 콜 모니터링 방법을 사용할 경우 이 암호화 기능을 액션 중 하나로 응용하여 사용할 수가 있다. 그리고 특히, 군사용으로 쓰일 경우에는 하드웨어 기반으로 암호화가 이루어져야 한다.

하드웨어를 사용하여 암호화를 구현하면 하드웨어에 암/복호화 전용 칩을 내장하는 방식 등을 사용하여 시스템에 부하를 주지 않고 빠른 속도의 암/복호화 연산을 할 수 있다. 또한 하드웨어에 키 값을 저장시키고 하드웨어를 분리시키거나 하드웨어 커버를 여는 순간 이 키 값을 사라지게 만들면 누군가 하드웨어 내의 키를 알아내기 위해 장비를 훔쳐간다 하더라도 키를 알아낼 방법이 없으므로 소프트웨어를 이용하는 방법보다 더 안전하다고 할 수 있다.

3.3 접근 제어

기존의 운영체계에서는 프로세스 및 파일들에 대한 아주 기본적이고 단순한 접근 제어 방식을 택하고 있었다. 예를 들어 UNIX 계열의 운영체계에서는 프로세스의 경우 단순히 프로세스의 소유자에 대한 권한만 강조하고, 파일의 경우 소유에 대한 권한은 소유자(owner), 그룹(group), 기타 다른 사용자(other) 등 세 가지로 나누고 접근 수 있는 권한의 종류에는 읽기, 쓰기, 실행, 권한없음 등 네 가지 종류가 있다. 이렇게 아주 기본적인 접근 제어 방식으로 인하여 공격자들이 쉽게 프로세스들을 제어하거나 또는 사용자의 등급을 쉽게 root까지 상향조정하여 시스템을 공격하고 있다. 따라서 프로세스 및 파일에 대한 접근 제어는 지금까지 연구되어져 온 여러 프로젝트에서 가장 많이 다루고 있는 중요한 부분이다. 접근 제어에 대한 대상은 크게 두 부분으로 나눌 수 있다. 하나는 사용자 레벨에서 커널 레벨로의 접근에 대한 제어이고, 다른 하나는 커널 레벨 사이에서의 접근에 대한 제어이다. 전자의 경우 대부분 시스템 콜을 사용한 사용자 레벨에서 커널 레벨로의 접근을 의미하고, 후자의 경우 프로세스간의 통신 또는 프로세스와 파일, 프로세스와 메모리 사이에서의 접근에 대한 제어를 의미한다. 보안 운영체계를 설계하기 위해서는 접근 제어의 두 가지 대상 모두를 포함하는 메커니즘을 제공하여야 한다.

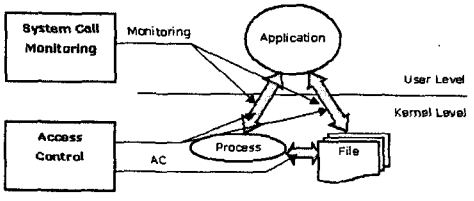
접근 제어 메커니즘뿐이니라 보안 운영체계에서 다루어져야 할 중요한 부분이 보안 정책이다. 보안 정책에서 가장 먼저 이루어져야 할 것은 초기 정책의 결정이다. 이 초기 정책은 두 가지의 고려사항이 존재하는데 하나는 완벽성(completeness)이고 다른 하나는 일치성(consistence)이다. 완벽성은 보안 정책이 과연 여러 공격에 대하여 올바르게 대처할 수 있느냐에 관한 관점이고 일치성은 여러 가지 정책들이 상호 간섭 없이 얼마나 일관성 있게 구성되어져 있느냐에 관한 관점이다. 초기 정책의 결정은 도구를 이용하여 자동으로, 그리고 관리자가 직접 수동으로 입력하여 이루어지기 때문에 반드시 앞서 고려한 완벽성과 일치성에 대한 점검이 필요하다. 따라서 보안 정책을 초기에 수립할 때 self-consistency 도구를 이용하여 반드시 점검하여야 한다.

완벽성과 일치성을 모두 만족시키는 초기 보안 정책을 가지고 접근 제어를 할 경우 반드시 추가해야 할 보안 정책이 생기게 된다. 이는 초기 정책이 아무리 훌륭하게 작성되었다 할지라도 보안 정책의 부재 및 오류로 인하여 로그를 보게 되면 접근 제어에 대하여 잘못 결정된 것을 확인할 수가 있고, 또한 결정을 하지 못하는 경우도 발생하기 때문이다. 따라서 새로이 보안 정책을 추가하는 경우에 대부분 관리자에 의해서 수동으로 이루어지는데, 이 경우 반드시 self-consistency 도구를 이용하여 기존에 존재하는 정책들과 일치성을 확인하여야 한다. 또한 최상위 레벨의 글로벌 보안 정책과 일맥상통하는지도 확인하여 올바르게 보안 정책이 추가될 수 있도록 하여야 한다.

이러한 전통적이고 기본적인 접근 제어 및 정책 이외에 추가적으로 확장할 수 있는 접근 제어 메커니즘에 대하여 살펴보면 다음과 같다.

Information Flow : 보안 운영체계에서 접근 제어와 보안 정책과 더불어 중요하게 다루어져야 할 사항은 정보의 흐름에 따르는 제어이다. 정보의 흐름에 따르는 제어는 앞서 설명한 접근 제어에 대한 보안 정책과는 다른 문제이다. 단순히 보안 정책을 기반으로 하는 접근 제어는 결정 사항이 바로 앞서 결정한 사항과 별개로 이벤트 별로 독립적으로 수행되어지는 것이다. 여기서는 이러한 독립적인 이벤트에 대한 제어가 아니라 어떤 이벤트들이 서로 연관되어서 순차적으로 진행하는 경우를 고려한다.

예를 들어 보안 측면에서 매우 중요한 파일의 경우 어떤 프로세스가 접근에 대한 요청을 하여 그 프로세스의 인증을 확인하고 보안 정책에 따라 정당한 요청일 경우 정책 결정 엔진은 읽기 또는 읽기/쓰기 접근을 허용할 것이다. 만약 이 파일이 ID와 password가 저장되어 있는 파일일 경우, 그 프로세스가 읽기 또는 읽기/쓰기 이외에 어떠한 행위를 하는지에 대해 관찰하여야 할 것이다. 그래서 모니터링을 통하여 그 파일이 인증되지 않은 외부로 유출이 되는지를 감시하여야 하고, 또한 암호화된 것을 풀어서 시스템이 제공하지 않는 다른 암호화 메커니즘을 이용하여 다시 암호화하는 등의 행위에 대하여 감시해야 할 것이다. 이러한 정책은 단지 파일을 읽기 허가 또는 쓰기 허가를 프로세스에 부여하는 접근 제어는 별개로 독립적인 상태 관리(state manage)를 통하여 수행되어져야 할 것이다. 그래서 실시간 모니터링을 통한 제어 또는 프로세스들의 trace pattern을 통한 제어로서 예측하지 못한 불법적인 상황이 발생한 경우 이를 발견하고 신속하게 대처하여야 할 것이다. 대처 방법으로 보안 측면에서 중요한 파일에 대해서 어떤 프로세스가 접근 요청할 경우 그 파일을 전부 제공하는 것이 아니라 필요한 서비스에 대하여 해당하는 부분만을 제공하거나 또는 허용된 프로그램 내에서만 서비스를 제공하는 등의 방법이 있다.



(그림 1) 접근 제어와 시스템 콜 모니터링을 통한 다양한 제어

Action : 기존에 연구된 프로젝트들에서의 접근 제어는 사용자 레벨에서 커널 레벨로의 접근뿐 아니라 커널 레벨에서 커널 레벨로의 접근도 고려하고 있다. 하지만 시스템 콜 모니터링 방법의 경우 사용자 레벨에서 커널 레벨로의 접근만을 다루고 있다(그림 1). 접근 제어에 대한 두 가지 경우에 따르는 가장 큰 차이는 접근 제어 결정에 따른 액션

부분이다. 기존 프로젝트들의 접근 제어는 단순히 서비스 요청의 허용 여부를 결정하는 한정된 액션을 갖는 반면 시스템 콜 모니터링은 서비스 요청의 허용여부 외에도 암호화 등 다양한 액션을 취할 수 있다. 예를 들어 프로세스가 권한이 없는 파일에 접근 요청을 하려 할 때, 시스템 콜 모니터링 방법으로는 이러한 서비스 요청을 제어할 수 없지만 접근 제어는 커널 내 모듈간의 서비스 요청도 모니터링하고 있기 때문에 제어가 가능하다.

IDS : 기존의 침입탐지 시스템은 커널과는 상관없이 커널 외부에서 시스템을 모니터링하고 있는 방식으로 비정상적인 접근을 일단은 허용했다가 나중에 히스토리를 보고 해킹 시나리오와 관련하여 침입을 탐지하는 수동적인 메커니즘이다. 이러한 침입탐지는 시스템 콜 모니터링으로 유사한 기능을 수행할 수가 있다. 시스템 콜을 모니터링하면서 실시간으로 커널로의 접근을 감시하면서 패턴 또는 히스토리를 참고로 비정상적인 접근이 탐지되면 즉각 접근을 막는 대처가 가능하다. 따라서 이미 비정상적인 접근을 허용한 후 나중에 침입을 탐지하는 침입탐지 시스템보다 미리 비정상적인 접근을 막아내는 시스템 콜 모니터링 방법으로 좀더 적극적이고 능동적인 방어를 수행할 수가 있다. 하지만 시스템 콜 모니터링 방법을 사용할 경우 처리 속도에 대한 성능이 떨어진다는 문제점이 있다.

3.4 감사

보안감사란 보안관련 행동에 관련된 정보의 인식, 기록, 저장, 분석을 포함하며 감사 레코드 결과는 어떤 보안관련 행동이 발생했으며, 누가 이에 대한 책임이 있는가를 결정할 때 활용될 수 있다. 감사는 보안감사 자동대응, 보안감사 데이터 생성, 보안감사 분석, 보안감사 검토, 보안감사 사건 선택, 보안감사 사건 저장 등의 기능을 포함하여야 한다.

먼저 보안감사 자동대응은 잠재적인 보안위반임을 암시하는 사건을 탐지했을 경우에 보안 정책에 따라 즉시 적절한 대응행동을 취하는 기능이다. 보안감사 데이터 생성은 보안관련 사건의 발생을 기록하는 기능이며 이것은 감사수준을 정의하고, 감사 기록될 사건들을 명세하며, 여러 형태의 감사 레코드 내에서 제공되어야 하는 최소한의 감사 정보들을 식별한다. 보안감사 분석은 예측되는 보안 위반이나 실제 보안 위반을 찾기 위해서 시스템 행동과 감사 데이터를 분석하는 자동화된 방법을 제공하는 것으로 이러한 분석은 급박한 보안 위반에 대한 침입탐지나 자동대응을 위해 수행될 수 있다. 보안감사 검토는 인가된 사용자가 감사 데이터 검토에 이용할 수 있는 감사 도구에 관한 것으로 보안감사 사건 선택은 감사대상에 대한 감사되어야 할 사건의 선택과 더불어 감사대상의 사건 집합으로부터 사건을 포함하거나 배제까지 포함한다. 마지막으로 보안감사 사건 저장은 안전한 감사 종적(audit trail)을 생성하고 유지할 수 있도록 하는 기능을 이야기한다.

4. 보안 운영체계를 위한 보안 구조

보안 구조는 크게 커널을 직접 수정하는 embedded 모델과 기존의 커널을 수정하지 않고 커널 위에 보안 구조를 삽입하는 loadable 모델이 있다. 이미 연구되어진 많은 프로젝트들의 초기에는 embedded 모델로 보안 구조를 운영체계에 포함하였다. 이러한 embedded 모델은 loadable 모델에 비해서 좀더 강력하고 처리 속도 대해서 좋은 성능을 나타내고 있지만 커널을 직접 수정하여야 하는 부담이 있고 UNIX나 FreeBSD 등과 같은 공개 소스 운영체계에서는 가능하지만 Microsoft에서 제공하는 상용 운영체계에서는 구현하기가 쉽지 않은 단점이 있다. 또한 운영체계의 버전이 높아져서 새롭게 배포될 경우 버전별로 보안 구조를 구성하여야 하는 부담도 존재한다. 이에 비해서 loadable 모델의 경우 운영체계의 기존 커널을 직접 수정하지 않고 보안 구조를 하나의 모듈로 구성하여 기존 커널 위에 삽입하기 때문에 embedded 모델에 비해서 구현하기가 용이하고 새로운 버전이 배포되어도 많은 수정을 가하지 않고 쉽게 재사용 할 수 있는 장점이 있다. 하지만 커널에서 직접 작업을 하지 않기 때문에 보안 정책의 강력한 제어와 속도 측면에서 embedded 모델에 비해서 낮은 성능을 나타낸다. Loadable 모델은 SELinux, LOMAC 등 근래의 프로젝트에서 주로 사용되는 구조이다.

전체적인 보안 구조와 함께 접근 제어에 대해 결정하고 집행하는 구조 또한 두 가지로 구분되어 지는데 하나는 커널 또는 접근 제어 모듈이 접근 제어에 대한 결정과 집행을 모두 담당하도록 하는 구조이고, 다른 하나는 결정과 집행을 나누어서 서로 독립된 모듈이 담당하도록 하는 구조이다. 보안 운영체계를 위한 프로젝트의 초기에는 결정과 집행을 커널 또는 접근 제어 모듈이 모두 담당하는 방법을 택하였으나, 요즘에는 결정과 집행을 독립된 모듈이 담당하도록 하는 구조를 택하고 있다. 이는 접근 제어를 결정하는데 필요한 보안 정책을 관리자가 필요에 따라 쉽게 수정하도록 하여 접근 제어에 대한 결정을 하는 모듈과는 독립적으로 집행을 담당하는 모듈은 결정에 따라 집행만 담당할 수 있도록 하여 좀더 유연한 접근 제어를 위한 것이다.

그리고 한가지 더 고려하여야 하는 사항은 다양한 보안 정책의 결정을 과연 운영체계의 어느 레벨에 두어야 하느냐에 대한 것이다. Single-level 정책 결정의 경우 단지 커널 레벨에서만 보안 정책에 따른 접근 제어 결정을 위한 엔진을 두고 수행할 경우 처리 속도에 대한 성능 면에서 좋을 것이고 어떠한 오버헤드도 발생하지 않을 것이다. 하지만 만약 기존에 수립된 보안 정책을 보고 접근 제어에 대한 결정을 내리지 못할 경우 정당한 접근 제어의 요청인 경우 허가를 하지 않고 거부할 수 있고, 또한 정당하지 못한 접근 제어의 요청에 대하여 허가를 할 수 있는 상황이 발생할 수 있다. 그리고 이러한 오류를 실시간에 관리자가 알 수가 없고 나중에 로그를 통하여 잘못된 것을 발견하여 보안 정책을 변경, 갱신 및 추가하여야 한다. 반면에 Multi-level 정책 결정의 경우 이러한 경우에 커널 레벨에서 결정하기 어려운 접근 제어에 대한 요청을 사용자 레벨로 올려서 관리자가 실시간으로 결정을 하여 올바른 접근 제어를 제공할 수 있고 즉시 보안 정책을 보완하여 나중에 동일한 요청에 대하여 커널 레벨에서 올바르게 결정을 내릴 수 있도록 할 수 있다. 하지만 Multi-level 정책 결정의 경우 Single-level 정책 결정보다 처리 속도에 대한 성능 면에서 떨어지고, 또한 사용자 레벨까지 올라가서 결정을 내려야하는 오버헤드가 발생하게 된다. 따라서 접근 제어 요청에 대한 결정을 Single-level로 할 것인지 또는 Multi-level로 할 것인지에 따라 처리 속도에 대한 성능 및 오버헤드와 실시간으로 올바르게 정책 결정을 내릴 수 있는 능력과의 상호 장단점이 존재한다.

5. 결 론

지금까지 연구된 보안 운영체계의 프로젝트들은 살펴보면 가장 중요한 이슈로 다루어진 부분은 접근 제어이다. 유연하고 강력한 접근 제어를 위해서는 운영체계의 사용자 레벨과 커널 레벨의 컴포넌트들은 모듈화 되어야 한다. 그래서 사용자 레벨에서 커널 레벨로의 통신, 그리고 커널 레벨 내에서의 통신 등 각 모듈간의 통신은 여러 통로가 아닌 하나의 통로만을 통해서 이루어지는 clean separation 되어야만 한다. 현재의 보안 운영체계의 프로젝트들은 대부분 기본적이고 전통적인 접근 제어 방식만을 고려하였으나 본 논문에서는 정보의 흐름 제어, 정책 결정에 따르는 다양한 액션 그리고 침입탐지 시스템 등의 확장된 접근 제어 메커니즘을 제안하였다. 또한 접근 제어 결정에 필요한 정책 수립에 있어서 완벽성과 일치성을 고려하여 올바른 정책을 구성하여 보안 운영체계의 설계 시 이러한 접근 제어 메커니즘을 중심으로 인증, 암호화, 감사 등의 메커니즘이 고려되어져야 한다. 그리고 운영체계의 특징에 맞도록 보안 구조를 embedded 또는 loadable 모델을 사용하여 커널에 삽입하여야 하고 접근 제어를 위한 결정과 집행의 모듈을 분리하여 유연성을 제공하도록 한다.

참고문헌

[1] Branstad. M, Tajalli. H, and Mayer. F, "Security issues of the Trusted Mach system," In Proceeding of the 4th Aerospace Computer Security Applications Conference 1988, Dec. 1988

[2] T. Fine and SE Minear, "Assuring Distributed Trusted Mach," In Proceeding of the 1993 IEEE Symposium on Research in Security and Privacy, May 1993

[3] Spencer E. Minear. "Providing Policy Control Over Object Operations in a Mach Based System," In Proceeding of the Fifth USENIX UNIX Security Symposium, June 1995

[4] R. Spencer et al. "The flask security architecture : System support for diverse security policies," In Proceeding of the 2000 USENIX Security Symposium, August 2000

[5] P. Loscocco and S. Smalley, "Integrating Flexible Support for Security Policies into the Linux Operating System," Technical report, NSA and NAI Labs, Oct. 2000

[6] N. Provos. Improving host security with system call policies. Technical Report 02-3, CITI, November 2002