유닉스 시스템에서 다양한 접근제어 정책을 이용한 커널 수준의 자동 암호화 기법

임 재 덕† · 유 준 석† · 김 정 녀††

요 약

현재까지 시스템의 보안을 위한 방법으로 다양한 접근제어 정책을 적용한 보안 커널들과 파일을 암호화하는 방법들이 연구되어 왔다. 보안 커널은 ACL, MAC, RBAC 등과 같은 접근제어 정책의 적용으로 시스템 내의 데이터에 대한 불법적인 접근을 효율적으로 통제 할 수 있는 방법을 제공할 수 있다. 하지만, 백업 매체나 디스크 자체의 도난에 대해서는 해당 매체 내에 저장되어 있는 데이터의 보호가 불가능하다. 접근제어 정책과는 별도로 파일을 암호화하여 저장하는 암호화 파일시스템에 대한 연구가 많이 소개되었지만, 접근제어 정책을 이용하여 상호 연동되는 구조를 제안한 사례는 거의 없다. 본 논문에서는 다양한 접근제어 정책을 가지는 보안 커널 내의 가상 파일시스템 수준에서 암호화 기능을 수행함으로써 사용자에 대해 투명성을 제공하는 암호화 파일시스템을 제안한다. 제안된 암호화 파일시스템은 접근제어 정책 중 MAC의 보안등급(class)에 기반한 암호키를 이용하기 때문에 비교적 간단한 암호키 관리 구조를 제공하고, 접근제어 정책 만을 적용할 때 존재할 수 있는 물리적 데이터 보안의 한계를 극복할 수 있다.

Automatic Encryption Method within Kernel Level using Various Access Control Policy in UNIX system

Jae-Deok Lim† · Joon-suk Yu† · Jeong-nyeo Kim††

ABSTRACT

Many studies have been done on secure kernel and encryption filesystem for system security. Secure kernel can protect user or system data from unauthorized and/or illegal accesses by applying various access control policy like ACL, MAC, RBAC and so on, but cannot protect user or system data from stealing backup media or disk itself. In addition to access control policy, there are many studies on encryption filesystem that encrypt file data within system level. However few studies have been done on combining access control policy and encryption filesystem. In this paper we proposed a new encryption filesystem that provides a transparency to the user by integrating encryption service into virtual filesystem layer within secure kernel that has various access control policies. Proposed encryption filesystem can provide a simple encryption key management architecture by using encryption keys based on classes of MAC policy and overcome a limit of physical data security of access control policy for stealing.

키워드 : 암호화 파일시스템(Encryption Filesystem), 접근제어(Access Control), 보안운영체제(Secure Operating System), 정보보호(Information Protection)

1. 서 론

최근 정부 조직이나 일반 기업들의 업무 환경이 LAN(Local Area Network)이나 지식관리 시스템(Knowledge Management System)과 같은 네트워크 기반으로 꾸준히 증가하고 있다. 네트워크 기반의 업무 환경은 시간과 인력 차원에서 많은 이점을 제공하지만, 네트워크 환경을 통해 조직 내외로 정보나 자료가 간단히 왕래 될 수 있고, 자료에 대한 접근 자체도 용이해져 기업이나 기관의 정보 노출의 가능성은 더욱 커지고 있다. 해킹 등에 의한 외부 침입도 증가하고 있지만, 최근에는 내부자에 의한 내부 침입이 더욱 더 증가하고 있다.

이에 대비하여, 파이어 월이나 침입탐지 시스템(Intrusion Detection system) 같은 응용 수준에서의 보안이 이루어지고 있지만, 이들은 각각 내부 침입자 및 새로운 형태의 침입에 대해 보안 상의 한계를 드러내고 있다. 그리고 시스템의 보안 패치 및 버전 업그레이드 등의 방법으로 시스템의 보안 취약점을 보완하고 있지만 시스템 차원의 보다 원천

† 정 회 원 : 한국전자통신연구원 연구원
†† 정 회 원 : 한국전자통신연구원 보안운영체제연구팀장
논문접수 : 2003년 1월 27일, 심사완료 : 2003년 5월 24일

적인 보안 기술이 요구된다. 이에 따라 최근 신분기반 혹은 자율적 접근제어(Discretionary Access Control, 이하 DAC), 강제적 접근제어(Mandatory Access Control, 이하 MAC), 역할기반 접근제어(Role Based Access Control, 이하 RBAC) 등과 같은 접근제어 정책을 적용한 보안 커널이 이슈가 되고 있으며, 그 기능의 정도에 따라 시스템의 보안등급을 평가하는 기준이 되기도 한다[1-4].

접근제어 정책은 주체와 객체의 관계에 따라 주체의 객체로의 접근 여부를 정의한 것이다. 접근제어 정책이 적용될 경우, 다양한 접근 정책과 엄격한 접근 규칙으로 인해 불법적인 사용자 혹은 비인가자의 시스템 자원으로의 허용되지 않은 접근을 효율적으로 차단할 수 있어 시스템 자원을 안전하게 보호할 수 있다. 본 논문에서 이용할 접근제어 정책은 MAC과 RBAC 정책이다[1-3].

MAC은 DAC에서 발생할 수 있는 문제 즉, 슈퍼 유저(루트 사용자)일 경우 DAC를 거치지 않는 문제를 해결할 수 있는 정책이다. MAC은 사용되는 주체와 객체의 보안등급과 범주(category)로 접근 권한을 검사한다[2]. 범주는 본 논문에서 무시한다. 주체의 등급이 객체의 등급보다 높은 경우 하위 등급의 객체에 대한 읽기가 가능하지만, 객체에 대한 쓰기는 주체의 등급과 객체의 등급이 같은 경우에만 가능하다. 이는 토로이안 목마 같은 침입을 방지할 수 있다[1].

RBAC은 역할에 기반 한 접근제어 정책으로써, 슈퍼 유저의 권한을 제한할 수 있다[3]. 이럴 경우 슈퍼 유저에게 모든 권한을 주었을 경우 발생할 수 있는 문제들을 해결할 수 있다. 또한 정해진 역할을 가진 사용자만이 접근할 수 있는 시스템 자원을 정하여 시스템 자원으로의 무분별한 접근을 막을 수 있다[1].

하지만, 접근제어 정책을 우회하여 임의의 시스템 자원을 접근할 수 있는 알려지지 않는 보안 취약점을 통해 접근이 가능할 수도 있고, 시스템 정보가 저장되어 있던 매체가 도난되었을 경우 저장 매체에 기록되어 있는 자료의 안전성은 보장할 수 없다. 예를 들어, 백업 관리를 맡은 사용자는 백업을 수행하기 위해 모든 데이터에 접근할 수 있어야 하고 이는 불법적인 데이터 유출의 가능성을 내포하고 있다. 이런 이유뿐만 아니라 평문 데이터를 저장 장치에 직접 저장하는 것은 많은 위험성을 가진다.

따라서 사용자의 데이터를 암호화하여 저장하는 방법에 많은 관심이 모아지고 있으며 많은 연구가 진행중이다. 데이터를 암호화하여 저장하는 방법은 시스템의 물리적인 보안보다 더 안전하며, 디스크 자체의 도난에 대해서도 중요한 데이터에 대한 유출을 방지할 수 있다. 지금까지 몇몇 대학과 연구기관에서 데이터를 암호화하는 파일시스템들이 연구되었다[5-8]. 하지만, 대부분의 암호화 파일시스템이 사용자 기반의 암호키를 이용하도록 구현되었기 때문에 사용자를 생성하고 삭제할 경우 해당 사용자에 대한 암호키의 관리가 필요하게 되고, 다중 사용자 시스템에서는 다른 사용자와 파일을 공유할 필요가 있을 때 해당 파일의 공유 문제도 발생한다. 본 논문에서는 유닉스 기반의 FreeBSD 시스템 내에 다양한 접근제어 정책이 구현된 보안 커널 내에서 동작하는 암호화 파일시스템을 제안한다[1]. 제안된 암호화 파일시스템은 [1]에서 구현된 MAC의 보안등급을 기반으로 생성된 암호키를 사용하기 때문에 사용자 기반의 암호키 관리 구조보다 간단한 암호키 관리 구조를 제공할 수 있으며, 임의의 보안등급을 가지는 암호화된 데이터를 해당 보안등급을 가지는 사용자에게 손쉽게 공유할 수 있다. 또한 암호화 파일시스템은 데이터의 암호화 저장이라는 특성을 이용하여 접근제어가 가지는 물리적 데이터 보안의 한계, 즉 평문으로 데이터가 저장되는 약점을 극복할 수 있는 수단을 제공한다.

2. 암호화 파일시스템

해킹 및 내부 침입자에 의한 자료 유출이 증가하고, 이에 따른 사용자 정보의 보안성에 대한 요구가 증가함에 따라 사용자의 데이터를 암호화하여 저장하는 방법에 많은 관심이 모아지고 있으며 많은 연구가 진행 중이다. 데이터를 암호화하여 저장하는 방법은 시스템의 물리적인 보안보다 더 안전하며, 디스크 자체의 도난에 대해서도 중요한 데이터에 대한 유출을 방지할 수 있다. 지금까지 몇몇 대학과 연구기관에서 데이터를 암호화하는 파일시스템들이 연구되었다[4-7]. 파일시스템 수준에서의 암호화는 사용자로 하여금 데이터 암호화를 위한 특별한 조작을 필요 없게 하여 사용상의 투명성을 제공하고, 시스템 침입자에 대해서 그들이 원하는 정보를 감춤으로써 데이터에 대한 보안성을 제공해 준다.

CFS(Cryptographic File System)는 NFS 파일시스템 내에 암호화 기능을 추가한 것으로, 암호화된 파일에 대해 표준 유닉스 파일시스템 인터페이스를 적용함으로써 시스템 수준에서의 보안 저장장치(secure storage)를 제공한다[5]. 사용자 영역에서 구현되었기 때문에 많은 문맥 교환이 발생하고, 이로 인해 시스템 성능에 한계가 있다.

TCFS(Transparent CFS)는 원격 NFS 서버와 통신하는 수정된 클라이언트 쪽의 NFS 커널 모듈이다[6]. 커널 영역에서 구현되어 암호화 서비스와 파일시스템 사이에 더 깊은 통합을 제공함으로써 CFS를 개선하였다. 사용자 암호키는 '/etc/tcfspasswd' 파일에 저장되는 데, 별도의 보호 메커니즘으로 보호하지 않는다면, 차후에 보안성을 감소시킨다.

Cryptfs는 Stackable Vnode Layer loadable kernel module로써 설계 및 구현된 파일시스템이다[7]. 클라이언트 파

일시스템을 '캡슐화' 함으로써 사용자에게 투명한 암호화 기능을 수행하며 커널 영역에서 동작한다. 암호키는 사용자 ID 뿐만 아니라 프로세스 세션 ID에 기반하고, 커널 메모리에 보관하므로 좀 더 강한 보안성을 제공하지만, 암호화를 수행하는 추가된 층(캡슐화)으로 인해 시스템에 과부하를 주고, 암호화된 파일을 다른 사용자들과 공유하는 것이 거의 불가능하다는 단점이 있다.

StegFS는 파일 암호화 기능에 steganography 특성을 제공하는 파일시스템이다[8]. 파일의 위치까지 은닉하여 좀 더 강한 보안성을 제공하지만, 파일 위치의 은닉성으로 인해 발생하는 데이터의 복사본 및 검색 횟수의 증가로 인해 성능 상에 많은 문제가 있다.

현재까지 개발된 파일시스템들은 대부분이 사용자 기반의 암호키를 이용하여 구현되었기 때문에, 다른 사용자로부터 자신의 파일을 보호할 수는 있지만, 사용자를 생성하고 삭제할 경우 해당 사용자에 대한 암호키의 관리가 필요하게 된다. 또한, 다중 사용자 시스템에서는 다른 사용자와 파일을 공유할 필요가 있을 때 해당 파일의 공유 문제도 발생한다. 본 논문에서는 가상 파일시스템 층에 암호화 기능을 통합하고, 보안 커널 내의 접근제어 정책을 이용함으로써 접근제어 정책이 가지는 물리적인 데이터 보안의 한계성을 극복하고, 기존 암호화 파일시스템이 가지고 있는 파일 공유의 문제를 해결하는 접근제어 기반의 암호화 파일시스템(Access Control based Encryption Filesystem, 이하 AEFS)을 제안한다. AEFS는 접근제어 정책을 이용하여 암호키에 대한 관리를 간단히 할 수 있다.

3. 보안 커널

AEFS가 구현 및 운영되는 시스템은 위에 설명된 다양한 접근제어 정책이 구현되어 있는 시스템이다[1]. 이 시스템에서 각각의 사용자는 보안 설정을 담당하는 보안관리자로부터 MAC 정책을 위해 수직적 개념의 보안등급과 수평적 개념의 범주를, RBAC 정책을 위해 역할(role)을 할당받는다. 보안관리자는 시스템관리자(루트 사용자)와 구분되는 사용자로써 시스템이 제공하는 '보안매니저' 역할을 가진 사용자이다. 기존의 시스템에서는 루트 사용자가 시스템 내의 어떠한 설정이라도 조작이 가능하지만, 이 시스템에서는 접근제어 정책 등과 같은 보안 관련 설정을 할 수 있는 보안관리자가 따로 존재하여 그 권한이 어느 정도 축소된다. 이는 최소 권한 분리의 개념이 적용된 것으로, 시스템 버그 등을 이용하여 임의로 루트 권한을 얻는 대부분의 시스템 공격에 효과적으로 대비할 수 있다. 물론 시스템 설치 초기에 루트 사용자에게 '보안매니저' 역할을 할당한다면 루트 사용자가 시스템관리자이면서 보안관리자도 될 수 있다. 이 경우라도 루트 사용자는 인증을 거칠 때 '보안매니저' 역할을 가지고 인증을 거쳐야만 보안관리자가 될 수 있다. 그렇지 않을 경우 '보안매니저' 역할을 가지고 있어야 조작이 가능한 작업은 수행할 수 없다. 관리자의 분리를 결정하는 것은 시스템을 운용하는 환경 및 정책에 따라 달라질 수 있다.

사용자는 시스템 인증시 자신에게 할당 받은 MAC, RBAC 정보의 범위 내에서 인증을 시도해야 하고, 인증을 시도할 때 입력한 값들은 MAC과 RBAC 정책을 적용하는데 사용된다. 기본값(보안등급 : 0, 범주 : 0, 역할 : 없음)으로 인증을 했을 경우, 기존 시스템의 일반 사용자와 같은 권한을 가지고, MAC과 RBAC 정책으로 접근이 제어된 객체로의 접근은 차단된다. 이 때, 루트 사용자라 할지라도 접근하려는 객체에 설정되어 있는 MAC과 RBAC 정보를 가지고 있지 않다면 그 객체에 접근할 수 없다. 이런 보안커널을 가지는 시스템은 보안등급, 범주 및 역할을 이용하여 시스템의 자원을 효율적으로 관리할 수 있다. 여기서, 보안등급을 가진 사용자가 취급하는 파일에 대해서는 좀 더 강화된 보안 기법이 적용될 필요가 있으며, 본 논문에서는 이를 위한 방법으로 보안등급을 가진 사용자가 생성하는 파일에 대해 커널 영역에서 자동으로 암호화하는 기법으로 AEFS를 제안한다.

4. 접근제어 기반의 암호화 파일시스템(Access Control based Encryption Filesystem, AEFS)

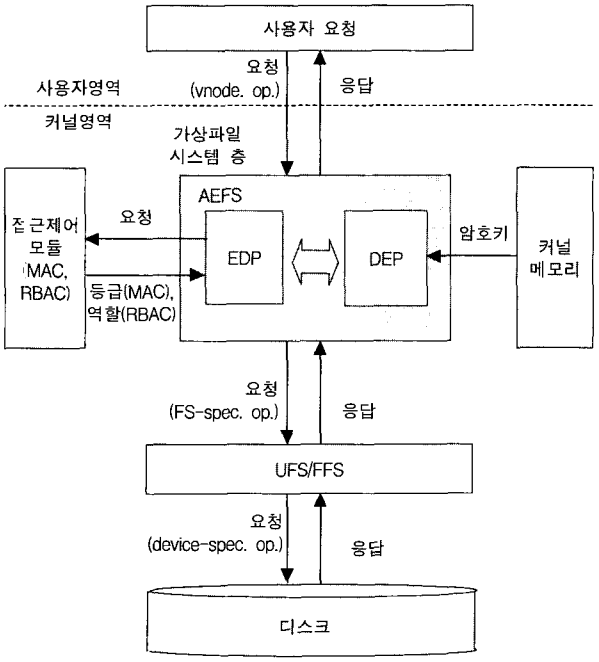
4.1 AEFS 구조

AEFS는 공개 운영체제인 FreeBSD 4.3 내의 가상 파일시스템 층에서 설계 및 구현되었으며, MAC과 RBAC 정책이 구현된 보안 커널과 연동되어 동작한다[1]. 암호화를 위한 별도의 계층을 가지지 않고 가상 파일시스템 층에서 통합되어 암호화 기능을 제공하기 때문에 별도의 암호화 층을 가질 때 발생되는 시스템 과부하를 줄일 수 있다[7].

AEFS는 시스템의 모든 파일을 암호화하지 않고, 보안등급을 가진 사용자가 생성한 파일에 한해서 암호화를 수행한다. 이는 보안등급을 가진 사용자가 생성하는 파일은 중요하다고 가정하여 사용자의 의식 없이 암호화 기능을 제공할 수 있고, 모든 파일에 대해 암호화를 수행할 때 발생되는 과부하를 최대한 줄일 수 있다. 또한 암호화 파일시스템에서 가장 중요하고 복잡한 암호키에 대한 관리도 간단히 할 수 있다.

AEFS의 암호키는 사용자 기반으로 제공되는 것이 아니고, MAC의 보안등급 기반으로 제공된다. 따라서, 같은 보

안등급을 가진 사용자는 같은 암호키를 파일 암호화에 사용하게 되므로, 접근제어 정책으로 다른 사용자의 파일 접근이 가능하다면 파일의 공유가 가능해 진다. 하지만, 암호키를 공유한다 하더라도 접근제어 정책으로 같은 등급의 사용자 접근을 제어할 수 있다. AEFS의 수행은 커널의 접근제어 정책이 통과했을 경우에만 이루어지기 때문이다. 암호키 또한 파일시스템 영역에서 커널 메모리로부터 얻어지므로 일반 사용자가 암호키를 얻는 것은 매우 어려운 일이다. AEFS는 가상 파일시스템 층에서 암호화가 이루어지므로 실제로 파일 입출력을 담당하는 UFS/FFS 같은 특정 파일시스템의 많은 부분을 수정하지 않고도 적용될 수 있다. (그림 1)은 MAC, RBAC 정책이 적용되어 운용되는 AEFS의 시스템 구조를 보여준다.

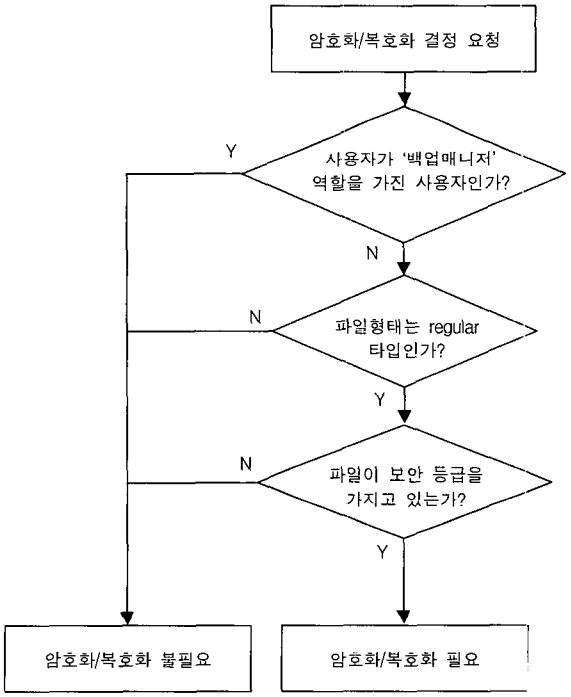


(그림 1) 접근제어 모듈이 적용된 AEFS 구조

AEFS는 크게 파일의 암호화를 결정하는 부분(Encryption Decision Part, 이하 EDP)과 암호화를 수행하는 부분(Data Encryption Part, 이하 DEP)으로 되어 있다. EDP는 사용자가 요청한 파일을 매개 변수로 받아 파일이 암호화/복호화가 필요한지를 결정한다. 만약 암호화/복호화가 필요하다면 DEP 부분으로 해당 데이터를 넘겨 암호화/복호화를 수행하도록 하고, 필요하지 않다면 사용자 요청을 바로 하위 레벨의 특정 파일시스템 연산으로 넘겨준다.

EDP는 요청된 파일의 암호화/복호화를 결정하기 위해 접근제어 모듈에 필요 정보를 요청한다. MAC 모듈은 해당 파일의 보안등급을, RBAC 모듈은 요청한 프로세스의 역할을 각각 EDP로 전달한다. EDP는 접근제어 모듈로부터 전달 받은 정보를 이용하여 해당 파일의 암호화/복호화 여부를 결정하며, (그림 2)에서 결정 과정을 보여준다.

첫 번째로 사용자가 시스템 자료들을 백업할 수 있는 '백업매니저' 역할을 가진 백업관리자인지 확인한다. 시스템 자료들을 백업을 하기 위해서는 시스템 내의 모든 파일에 접근할 수 있어야 하며, 이는 접근제어 정책을 적용 받지 않아야 한다. 이 때 백업관리자가 악의적인 목적을 가지고 있다면 시스템 내의 어떤 파일에 대해서도 내용을 볼 수 있거나 외부로 유출할 수 있다. 이런 위험을 방지하기 위해서 파일에 접근하는 사용자의 역할을 확인하여 '백업매니저' 역할을 가지고 있을 경우 암호화/복호화 기능을 제공하지 않는다. 따라서 파일이 암호화되어 있을 경우 해당 파일이 복호화되지 않고 암호화된 형태로만 읽혀지게 되어 암호화된 파일을 보호할 수 있다. 또한 백업관리자에 의한 파일 훼손의 위험은 사용자가 '백업매니저' 역할을 가질 경우 읽기만 가능하도록 RBAC 정책을 설정하여 막을 수 있다. 두 번째로 암호화/복호화는 현재 정규 파일 형태일 경우에만 제공하도록 제한하였다. 세 번째는 보안등급을 가진 파일에 대해서만 암호화/복호화를 제공한다. 이는 앞서 설명했듯이 파일시스템의 과부하를 줄일 수 있고, 중요한 파일에 대한 기밀성을 보장할 수 있다.



(그림 2) EDP에서의 암호화 결정 과정

DEP에서는 OFB(Output FeedBack) 모드의 Blowfish 알고리즘이 사용된다. Blowfish 알고리즘은 DES나 IDEA 등의 타 알고리즘과 비교하여 안전성 측면에 문제가 없으면

서도 연산속도나 자원의 활용도 측면에서 좋은 특성을 지녔다. 또한 구조적 분석의 위험성을 줄이기 위해 OFB 모드를 이용한다[9-10]. OFB 모드의 적용은 시스템의 성능을 고려하여, 파일의 전체 부분에 대해 적용하는 것이 아니라 디스크에 저장되는 페이지 단위(4KB)로 적용하도록 한다. 블록 암호 알고리즘들은 그 구조가 비슷하고 유사한 메커니즘으로 동작하기 때문에 수월하게 다른 알고리즘을 적용할 수 있다.

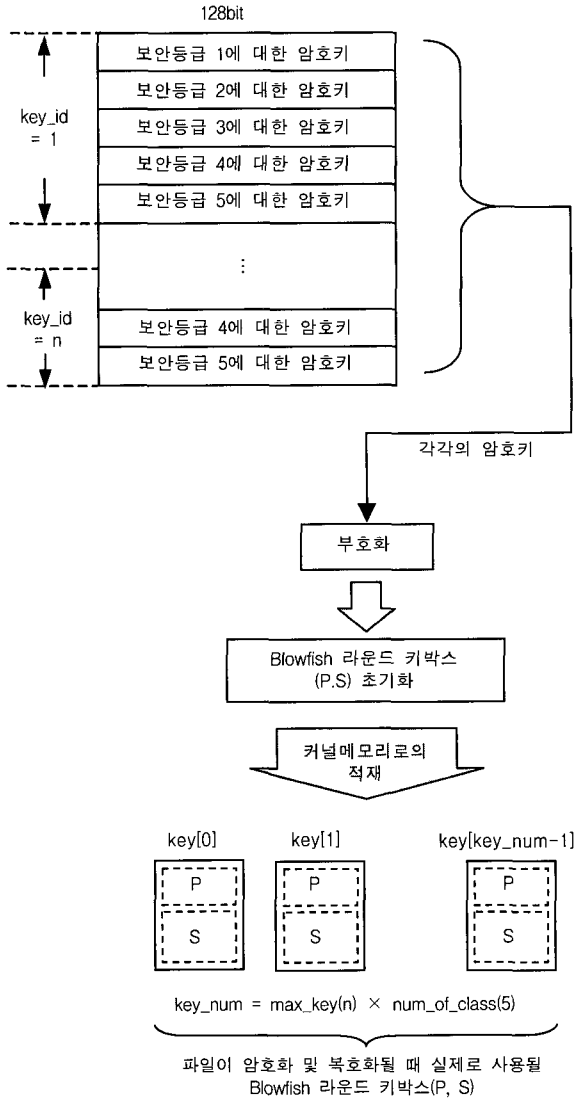
4.2 암호키 관리

AEFS의 암호키는 사용자 기반이 아닌 MAC의 보안등급에 기반하여 관리된다[1]. 시스템에서 보안등급을 가지는 이유 중 하나는 높은 보안등급의 자료를 낮은 사용자가 열람하지 못하도록 하는데 있다. 따라서 보안등급이 다른 사용자들간에는 파일을 공유할 필요가 없다. 보안등급 기반의 암호키 사용은 사용자 기반의 암호키 사용시 발생하는 파일 공유 문제를 효과적으로 해결할 수 있다. 같은 보안등급을 가지는 사용자들의 파일은 서로 공유될 수 있기 때문이다. 하지만, 접근제어 정책을 적용하여 같은 보안등급의 다른 사용자로부터 자신의 파일을 보호할 수도 있다. AEFS가 구현된 보안 커널에서는 1에서 5까지의 보안등급을 제공하며, 보안등급 '5'는 최고 보안등급이 된다[1]. 기본값은 '0'의 보안등급으로 이 값으로 인증을 시도하였을 경우에는 보안등급이 없는 보통 사용자로 인식한다. 사용자는 시스템에서 제공하는 보안등급 범위 내에서 적절한 범위를 할당받을 수 있고, 할당 받은 보안등급 범위 내에서 하나의 보안등급으로 시스템에 인증을 시도할 수 있다. 인증을 시도할 때 입력된 보안등급은 인증 이후 시스템 내에서의 사용자 보안등급이 되고, 파일에 접근할 때 MAC 정책에 사용된다. 또한 EDP에서 암호화/복호화의 필요 여부를 판단할 때 사용되며, 처음 파일이 생성되어 암호화될 때 커널 메모리로부터 해당 암호키를 할당 받는 기준이 된다.

암호키는 '보안매니저' 역할을 가지는 사용자인 보안관리자 만이 생성할 수 있으며 생성된 암호키는 특정한 파일에 보관되고, 이 파일은 보안관리자 만이 접근할 수 있는 디렉토리('/sos')에 저장된다.

한 번의 생성으로 각 보안등급 별로 128-bit의 키가 생성되어 보관되고, 생성된 각각의 보안등급별 암호키는 갱신된 횟수를 의미하는 key_id를 가진다. 즉, 처음 생성될 경우 암호키는 1이라는 key_id를 가지고, 추가로 생성될 수도 있으며, 이때는 key_id 값이 하나 증가한다. 이전에 생성된 암호키는 새로운 암호키가 생성되기 전에 사용된 파일들의 복호화에 사용된다.

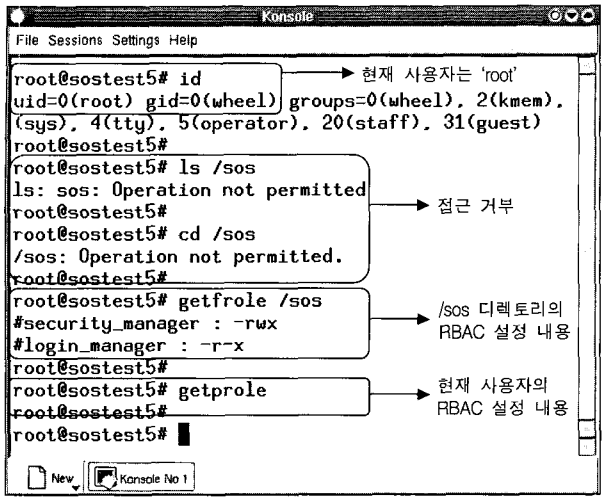
Key_id는 시스템이 사용하는 각 보안등급 별 암호키의 종류를 다양화하여 암호키에 대한 보안을 강화할 수 있다. 암호키 초기화는 (그림 3)에서 보는 바와 같이 커널이 부팅될 때 이루어진다. 암호키파일에서 각 key_id 별로 각 보안등급에 해당하는 128bit 암호키를 차례로 읽어 들여 부호화를 거친 후 각 암호키에 해당하는 라운드 키박스를 초기화하여 커널 메모리에 배열 형태로 차례로 적재한다. 배열 형태로 커널 메모리에 적재된 각 보안등급에 해당하는 암호키의 라운드 키박스는 실제 파일을 읽고 쓸 때 파일에 저장된 암호키의 정보로부터 간단한 계산을 통해 쉽게 찾아질 수 있다.



(그림 3) 암호키의 초기화 과정

암호키에 해당하는 라운드 키박스의 배열 위치를 찾는 정보는 파일이 생성될 때의 사용자 보안등급 즉, 파일의 보안등급과 사용된 암호키의 key_id가 된다. 이 값들은 디스크 메타 정보 구조체인 inode의 사용되지 않는 필드에 각

각 저장된다. 단기간 내에 128-bit 암호키 해독은 이론상으로 불가능하고, 오랜 기간이 경과된 후 해독이 가능하다 하더라도 사용되는 암호키가 임의의 주기로 갱신되므로 해독은 거의 불가능하다고 볼 수 있다.



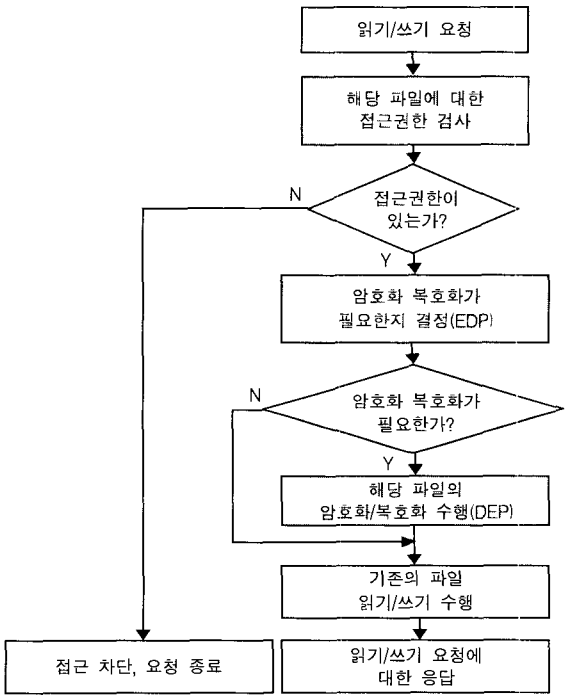
(그림 4) 키파일이 저장되어 있는 디렉토리 보안 설정

암호키가 파일에 저장되어 있는 것은 어느 정도의 위험성을 가지고 있으나, 접근제어 정책으로 보호되어 있으며, 파일에 저장된 형태 그대로 커널 메모리에 적재되는 것이 아니라 커널 내부에서의 자체적인 부호화 과정을 거쳐 적재되므로 안전성을 보장할 수 있다. (그림 4)는 암호키와 관련된 파일이 저장되어 있는 '/sos' 디렉토리를 RBAC 정책을 이용하여 보호하는 것을 보여 준다. RBAC 정책은 앞의 3절에서 언급했듯이 사용자가 가진 역할과 객체, 즉 디렉토리나 파일에 설정된 역할의 관계에 따라 접근 여부를 결정하는 것이다. (그림 4)에서 소개되는 'getfrole'은 파일에 설정되어 있는 역할정보를, 'getprole'은 현재 프로세스(사용자)가 가지고 있는 역할정보를 확인하는 명령어로 보안커널에서 RBAC 정책을 위해 제공하는 유틸리티들이다[1]. 현재 '/sos' 디렉토리는 '보안매니저(#security_manager)'와 '로그인매니저(#login_manager)' 역할을 가진 사용자만이 접근(x) 및 읽기(r)가 가능하고, 쓰기(w)는 '보안매니저(#security_manager)' 역할을 가진 사용자 만이 가능하도록 설정되어 있다. 따라서, 임의의 사용자가 이 두 역할 중 어느 하나의 역할도 가지고 있지 않다면 해당 디렉토리에 접근조차 할 수 없다. (그림 4)에서는 루트 권한을 가진 사용자가 '/sos' 디렉토리를 접근하고 있지만, '/sos' 디렉토리에 설정된 역할('보안매니저(#security_manager)', '로그인매니저(#login_manager)')이 없으므로 해당 디렉토리로의 접근이 불가능하다. 또한 시스템 해킹을 통해 얻어지는 루트 권한은 거의 시스템 버그 등을 이용하여 루트의 아이디('0')를 얻는 것이지만, 각 사용자가 가지는 역할들은 정상적인 인증 과정을 통해서만 얻어질 수 있으므로 시스템 해킹을 통해서 단순히 어떤 사용자의 아이디를 얻는 방법으로는 시스템에 사용되는 역할을 얻을 수 없다.

4.3 파일의 읽기/쓰기 과정

파일에 대해 읽기 및 쓰기 작업이 수행되기 전에, 시스템에서는 접근제어 모듈을 통해 접근을 시도하는 주체 즉, 프로세스가 접근하려는 파일에 대해 읽기 혹은 쓰기 권한이 있는지를 검사한다. 접근제어 모듈에서 주체의 객체에 대한 접근을 허가할 경우 파일에 대해 읽기 및 쓰기 연산이 수행된다. 기본적인 읽기 및 쓰기 과정은 Cryptfs의 방법을 기반으로 한다[7].

(그림 5)는 파일의 읽기 및 쓰기 과정 중 접근제어 정책과 EDP, DEP가 어떻게 적용되는지 보여 준다.



(그림 5) 파일 읽기 및 쓰기 과정

응용 프로그램이 파일 읽기를 요청하면 EDP는 파일의 등급과 사용자 역할을 통해 파일을 복호화할 지를 결정하고, 복호화할 필요가 없다면 바로 하위 레벨의 특정 파일시스템으로 요청을 전달하고, 읽은 데이터를 응용 프로그램으로 전달한다. 복호화해야 한다면, 커널 메모리로부터 해당 파일의 암호키에 대한 라운드 키를 얻고, 요청된 영역을 페이지 단위의 경계까지 영역을 확장한다. 복호화를 하기 전에 확장된 영역으로 복호화할 데이터를 특정 파일시스템으로부터 읽어온 다음, 획득한 키를 이용하여 암호화된 데이

터를 복호화한다. 이 후 복호화된 데이터 중 요청된 영역만을 응용프로그램으로 전달한다. 데이터 영역의 확장은 파일이 페이지 단위로 OFB 모드를 이용하여 암호화되었기 때문이다[10].

파일의 쓰기 과정은 읽기 과정보다 복잡하다. 파일 쓰기가 요청되면, EDP는 파일의 등급과 사용자의 역할에 따라 암호화를 결정한다. 파일이 보안등급을 가지고 있지 않거나, 사용자가 '백업매니저' 역할을 가지고 있거나, 혹은 파일이 처음 생성될 때 사용자가 보안등급을 가지고 있지 않을 때는 암호화가 수행되지 않는다. 암호화가 필요없다면, 파일 쓰기 요청은 하위 레벨의 특정 파일시스템으로 전달되고, 암호화과정 없이 파일이 쓰여진다.

암호화가 필요하다면, 파일이 수정될 경우 파일 보안등급에 해당하는 암호키를, 파일이 생성될 경우 사용자의 보안등급이 해당하는 암호키를 커널 메모리로부터 얻어온다. 파일은 OFB 모드에 의해 암호화되어 있으므로, 한 페이지 내에서 임의의 데이터는 이전 데이터와 관련되어 있다. 따라서, 요청된 영역의 경계가 속한 페이지 전체를 읽어와 복호화하고, 쓰여지지 않는 나머지 영역을 차후의 암호화를 위해 보존한다.

그리고, 쓰여질 데이터를 이전의 복호화 과정에서 보존하고 있던 부분과 합쳐 완전한 페이지를 형성한다. 지금까지는 암호화되기 전의 평문 데이터이므로, 형성된 페이지를 해당 암호키를 이용하여 암호화하고, 데이터를 저장하기 위해 하위 레벨의 특정 파일시스템으로 암호화된 데이터를 전달한다.

4.4 파일 재암호화

파일의 보안등급이 변경되는 경우, 변경된 등급에 해당하는 암호키로 파일을 다시 암호화해 주어야 한다. 또한 파일 정보 구조체 내의 보안등급 역시 변경된 보안등급으로 변경해 주어야 한다. 재암호화 과정은 일단 기존의 보안등급 암호키로 해당 파일을 복호화한 후 변경된 보안등급 암호키로 암호화한다. 파일 전체를 복호화한 후 암호화하는 것이 아니라, 커널 내에서 입/출력되는 단위 크기 즉, 페이지 단위 만큼 위의 과정이 반복된다. 이 때, 변경된 보안등급의 암호키는 가장 최신의 key_id에 해당하는 보안등급 암호키를 적용시킨다.

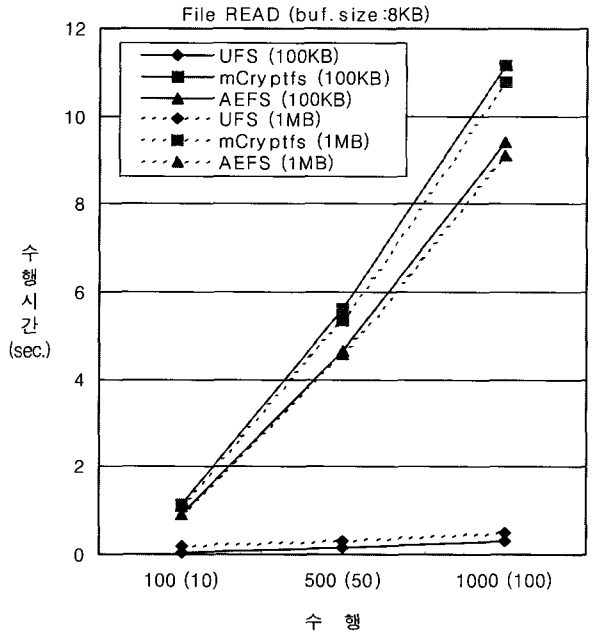
5. 실험 및 결과

성능 측정을 위한 실험은 암호화를 적용하지 않았을 경우(UFS)와 추가된 암호화 층을 이용한 경우 즉, 기존의 Cryptfs를 본 논문의 설계와 같이 수정한 경우(mCryptfs)와 본 논문에서 구현한 가상파일시스템 층에서의 직접적인 암호화를 제공할 경우(AEFS)에서 각각 이루어진다. 비교 시스템으로 Cryptfs를 선정한 이유는 기존에 연구된 유닉스 기반의 파일시스템 중 성능이 비교적 가장 좋기 때문이다[7]. 또한 Cryptfs 구조를 그대로 적용하지 않고, AEFS와 유사한 구조로 변경한 이유는 Cryptfs가 사용자 기반의 암호키를 기반으로 하였기에 AEFS에서의 암호키 접근 방법에서 차이가 있기 때문이며, 접근제어 참조부분을 제외한 부분에 있어서 유사한 환경을 제공하기 위함이다. 본 실험은 펜티엄 1.4GHz CPU와 256MB의 메인 메모리를 가진 시스템에서 수행되었으며, 캐싱에 의한 파일 입출력 영향을 줄이기 위해 각 실험 수행 전 캐싱의 영향을 제거하였다.

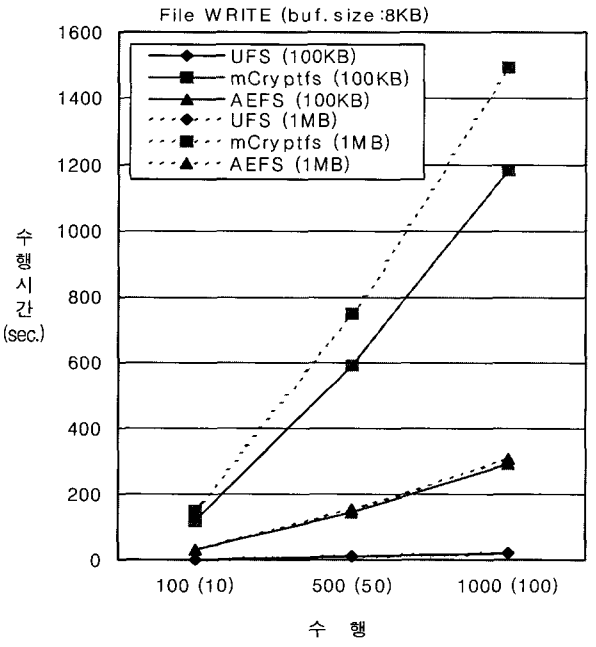
(그림 6)은 파일에 대한 읽기 및 쓰기 성능을 보여주며, 각각 100KB, 1MB 크기의 파일에 대해 그래프에 표시된 수행 횟수만큼 시험을 거쳐 수행되었으며, 결과는 각 수행 횟수만큼 테스트를 실시했을 때의 누적된 시간이다. 수행횟수의 괄호부분은 1MB 크기의 파일에 대한 수행횟수를 가리킨다. 암호화를 수행한 경우는 암호화 알고리즘 자체의 과부하로 인해 암호화를 적용하지 않았을 경우보다 성능이 떨어지는 것을 볼 수 있다. 특히 전체적으로 보면 읽기에서보다 쓰기에서 성능이 많이 떨어짐을 볼 수 있다. 이는 4.3절에서 설명한 것과 같이 AEFS와 Cryptfs는 데이터를 암호화할 때 데이터의 안정성을 위해 체인 모드(OFB 모드)를 사용하기 때문이다. 따라서 쓰기 연산이 수행될 경우 데이터의 수정을 위해 수정될 부분의 암호화된 데이터를 복호화해야 하고, 이는 파일시스템의 읽기 연산을 필요로 한다. 따라서, 데이터가 많으면 많을수록 읽기 및 쓰기 연산의 중복이 불가피해진다. 또한 디스크 I/O 특성상 일반 파일시스템의 쓰기 성능이 읽기 성능보다 많이 떨어진다. 따라서, 암호화 파일시스템의 쓰기 성능이 읽기 성능보다 더욱 많이 떨어진다.

그러나, 암호화를 위한 별도의 계층을 가지는 암호화 방법과 비교해 볼 때 AEFS의 성능이 (그림 6)에서 보는 것처럼 읽기와 쓰기 연산 성능이 모두 우수함을 볼 수 있다. 특히 AEFS가 기존 암호화 파일시스템과 비교하여 읽기 성능에서 보다 쓰기 성능에서 월등히 우수하다. 이는 AEFS가 별도의 암호화를 위한 층을 유지할 때 발생되는 과부하를 가지지 않기 때문이다. Cryptfs에서는 암호화를 위한 암호화 기능이 파일시스템에 통합되어 있지 않고, 별도의 암호화 층을 가지고 있다. 따라서 암호화 및 복호화를 위해 암호화 층에 추가적인 데이터의 복사가 이루어져야 하고, 모든 파일시스템 연산이 암호화 층을 거쳐서 아래 파일시스템 층으로 전달된다. 즉, Cryptfs에서는 쓰기 연산에서 수행되는 추가적인 읽기 연산과 더불어 수행하지 않아도 되는 연산들이 AEFS보다 상대적으로 많아지기 때문에 AEFS보다 성능

이 많이 떨어진다. (그림 6)의 그래프에서 보면 전체적으로 다루어진 파일의 크기가 10M~100M이고 크기가 작을수록 성능의 차이가 그렇게 크지 않음을 알 수 있다. 이는 시스템에서 자주 사용되는 파일의 크기가 대부분 작다는 것을 감안하면 사용자는 시스템의 과부하를 그렇게 크게 느끼지 않을 수 있다는 것을 의미한다. 하지만, 발생되는 과부하가 기존의 모든 암호화 파일시스템에서도 그렇듯이 시스템 자체 구조에 의한 과부하보다 암호화 및 복호화에 따른 과부하가 대부분이며 이 부분은 현재 암호화를 적용하는 어떤 시스템에서도 고민 대상이 되고 있다. 이 부분에 대한 성능 향상은 차후 하드웨어 기반의 암호화 가속기 등의 사용을 통해 극복되어져야 할 것이다.



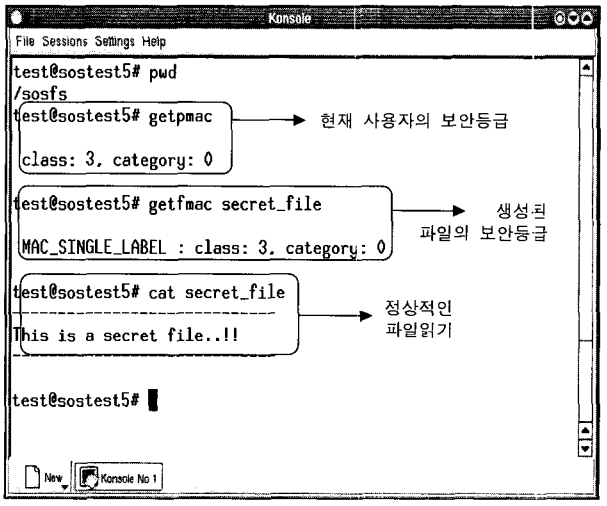
(a) 파일 읽기 성능



(b) 파일 쓰기 성능

(그림 6) 파일의 읽기 및 쓰기 성능

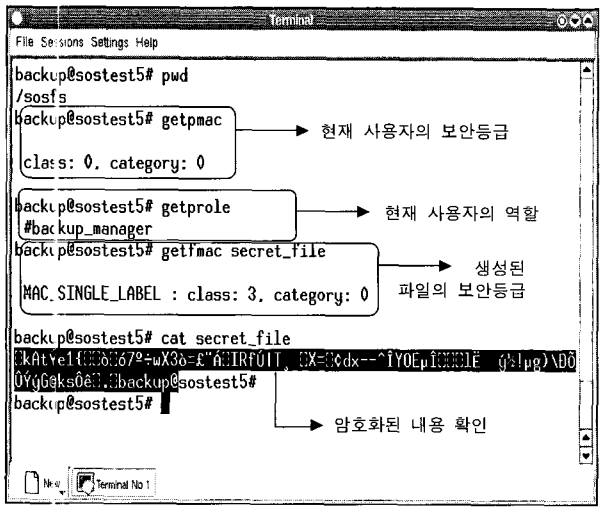
(그림 7)은 3등급의 보안등급을 가진 사용자가 파일을 생성하는 경우 해당 파일이 자동으로 암호화되는 것을 보여주고, 사용자가 생성된 파일을 읽을 때 정상으로 읽을 수 있음을 보여준다. 이는 커널 수준에서 파일을 생성하는 사용자의 보안등급을 자동으로 인지하여 파일 암호화가 진행되므로 사용자는 암호화에 따른 특별한 조작을 하지 않아도 파일이 자동을 암호화되어 저장된다. 'getpmac'은 현재 프로세스(사용자)의 MAC 정보 값을, 'getfmac'은 파일에 설정되어 있는 MAC 정보 값을 확인하는 명령어로 보안커널에서 MAC 정책을 위해 제공하는 유틸리티들이다[1]. 이들 명령어를 통해 해당 사용자 및 객체의 보안등급과 설정된 보안 범주들의 정보를 볼 수 있다.



(그림 7) 보안등급 3을 가진 사용자가 파일을 생성하는 화면

(그림 8)은 (그림 7)에서 생성된 파일이 암호화되어 있는지를 확인하는 그림으로 '백업매니저(#backup_manager)' 역할을 가진 사용자로 실제 위의 파일이 암호화되어 있는지 확인할 수 있다. '백업매니저' 역할을 가진 사용자는 시스템의 데이터를 백업하기 위해 모든 데이터로의 접근이 가능해야 하고 이는 악의적인 의도를 가졌을 경우 정보 유출의 위험성을 가지게 된다. 이런 정보 유출의 위험성을 방지하기 위해 '백업매니저' 역할을 가진 사용자에 대해서는 암호화 기능이 제공되지 않기 때문에 파일이 저장된 형태 그대로를 읽는다. 따라서 이 역할을 가진 사용자를 통해 해당 파일의 암호화 여부를 확인할 수 있다. (그림 8)에서 사용자는 보안등급을 가지고 있지 않지만, '백업관리자' 역할을

가지고 있으므로 (그림 7)에서 생성된 3등급의 파일을 접근할 수 있지만, 디스크에 저장된 형태 그대로를 읽는다. 즉, 파일이 암호화되어 저장되어 있음을 확인할 수 있다.



(그림 8) '백업매니저' 역할을 가진 사용자가 암호화된 파일을 확인하는 화면

6. 결론 및 향후과제

본 논문에서는 접근제어 정책을 이용하여 시스템 운용에 큰 과부하를 발생시키지 않으면서, 사용자들간의 손쉬운 데이터 공유와 간단한 암호키 관리를 제공하는 가상 파일시스템 층에 통합된 접근제어 기반의 암호화 파일시스템(AEFS)의 구조를 제안하였다. 이를 통해 접근제어 정책 만을 적용하였을 경우 발생할 수 있는 문제 즉, 백업 관리자의 사용자 데이터 열람 가능, 백업 매체 도난 및 접근제어를 우회한 데이터 접근 등의 문제들을 해결할 수 있다.

데이터의 암호화는 시스템의 물리적인 보안을 이룰 수 있다는 점에서 꼭 필요한 방법이다. 하지만, 암호화 및 복호화에 따른 과부하가 현재 암호화를 적용하는 어떤 시스템에서도 고민 대상이 되고 있고, 제안된 암호화 파일시스템이 보다 큰 크기의 파일에 적용될 경우 이 부분에 대한 성능 향상은 차후 하드웨어 기반의 암호화 가속기 등의 사용을 통해 극복되어져야 할 것이다. 또한, AEFS는 성능 상의 기선 사항 보다는 최근 부각되고 있는 접근제어 정책이 적용된 보안커널 내에서 발생할 수 있는 보안 취약점을 보완하고, 중요한 데이터를 좀 더 안전하게 저장함을 제공할 수 있으며, 기존의 복잡한 암호키 관리 및 파일 공유 문제를 개선하였다는 데 의의가 있다고 할 수 있다.

AEFS에서 보완되어야 할 것은 암호키의 보호 방법으로써, 접근제어 정책으로 암호키 파일에 대한 임의의 접근을 막고는 있지만 시스템 내부에 저장하는 것은 향후 보안 취약점으로 남을 것이다. 따라서 스마트 카드 등의 외부 장치의 이용으로 암호키 파일의 내용을 시스템에 저장하지 않는 방법들이 연구되어야 할 것이다.

참 고 문 헌

[1] J. G. Ko, J. N. Kim & K. I. Jeong, "Access Control for Secure FreeBSD Operating System," Proc. of WISA2001, The Second International Workshop on Information Security Applications, Seoul, KOREA, pp.247-254, Sep., 2001.

[2] Bell, David Elliott & Leonard J. La Padula, "Secure computer system : Unified exposition and multics interpretation," MITRE Technical Report 2997, MITRE Corp, Bedford, MA, 1975.

[3] David F. Ferraiolo, Ravi Sandu & Serban Gavrila, "A Proposed Standard for Role-Based Access Control," ACM transaction on Information and System Security, Vol.4, No.3, pp.224-274, Aug., 2001.

[4] "Common Criteria for Information Technology Security Evaluation, Part 2 : Security functional requirements," Version 2.1, 1999.

[5] M. Blaze, "A Cryptographic File System for Unix," Proc. of the first ACM Conference on Computer and Communications Security, Fairfax, VA, Nov., 1993.

[6] G. Cattaneo & G. Persiano, "Design and Implementation of a Transparent Cryptographic File System for Unix," Unpublished Technical Report. Dip. Informatica ed Appl, Universita di Salerno, July, 1997.

[7] E. Zadok, I. Badulescu & A. Shender, "Cryptfs : A Stackable Vnode Level Encryption File System," Technical Report CUCS-021-98, Computer Science Department, Columbia University, July, 1998.

[8] Andrew D. McDonald & Markus G. Kuhn, "StegFS : A Steganographic File System for Linux," IH '99 LNCS 1768, pp.463-477, 2000.

[9] B. Schneier, "Blowfish : In Applied Cryptography," 2nd Ed., John Wiley & Sons, pp.336-339, 1996.

[10] B. Schneier, "Algorithm Types and Modes : In Applied Cryptography," 2nd Ed. John Wiley & Sons, pp.189-197, 1996.

임 재 덕

e-mail : jdscol92@etri.re.kr

1999년 경북대학교 전자공학과(학사)

2001년 경북대학교 대학원 전자공학과 (석사)

2000년~현재 한국전자통신연구원 연구원

관심분야 : 시스템 보안, 네트워크 보안, 운영체제, 병렬 처리

유 준 석

e-mail : jsyu92@etri.re.kr

1999년 성균관대학교 정보공학과(학사)

2001년 성균관대학교 대학원 전기전자 및 컴퓨터 공학부(공학석사)

2001년~현재 한국전자통신연구원 연구원

관심분야 : 암호이론, 보안운영체제, Mobile IPv6 등

김 정 녀

e-mail : jnkim@etri.re.kr

1987년 전남대학교 전산통계학과(학사)

1995년~1996년 Open Software Foundation Research Institute 파견 (공동연구)

2000년 충남대학교 대학원 컴퓨터공학과 (석사)

1988년~현재 한국전자통신연구원 보안운영체제연구팀장

관심분야 : 운영체제, 시스템 보안, 네트워크 보안, 분산 처리