접근제어모델을 이용한 암호화파일시스템의 설계

임재덕*, 은성경*, 김정녀*
*한국전자통신연구원 보안운영체제연구팀
{jdscol92, skun, jnkim}@etri.re.kr

Design of Crypto-Filesystem using Access Control Model

Jae-Deok Lim*, Sung-Kyung Un* and Jeong-Nyeo Kim*
*Electronics and Telecommunications Research Institute(ETRI)

요 약

현재까지 시스템의 보안을 위한 방법으로 다양한 접근제어 모델과 파일을 암호화하는 방법들이 연구되어 왔다. 접근제어 모델는 데이터에 대한 불법적인 접근을 효율적으로 통제 할 수 있는 방법을 제공하지만, 슈퍼 유저 및 백업 관리자 등과 같이 특수한 권한을 가진 사용자에 대해서 그리고 백업 매체나 디스크 자체의 도난에 대해서 사용자 데이터의 보호가 불가능하였다. 반면, 데이터를 암호화하는 기법은 사용자 데이터를 보호할 수 있지만, 각 사용자 간의 데이터 공유 문제 및 데이터 암호화로 인해 발생하는 시스템의 과부하를 발생시킨다. 본 논문에서는 암호화 파일시스템에 접근제어 모듈을 적용시킴으로써 암호화 파일시스템의 구조를 간단히 하고, 사용자 간에 데이터를 공유할 수 있는 구조를 제안한다.

1. 서론

네트워크의 발전과 개방성으로 인해 시스템의 중요한 정보에 대한 내부 혹은 외부 침입자에 의한 불법적인 사용이나 위협이 커지고 있다. 따라서, 시스템의 보안을 위한 많은 연구가 이루어지고 있으며, 대표적인 것으로 IDS 와 파이어 월 같은 것이 있지만 한계가 있다. IDS 는 새로운 방법의 침입에는 효과가 없으며, 파이어 월은 이미 침입자가 시스템 내부에 침입했다면 아무 소용이 없다[1]. 또한 시스템의 보안 패치 및 버전 업그레이드 등의 방법이 사용되고 있지만, 시스템 차원의 보다 원천적인 보안 기술이 요구된다. 시스템 자원에 대한 접근 통제를 강화하는 접근제어 모델과, 시스템에 저장되는 데이터에 대한 데이터 암호화 모델이 시스템 보안의 예가 될 수 있다.

최근 접근제어 모델을 적용한 보안 커널이 이슈가 되고 있으며, 그 기능의 정도에 따라 시스템 보안 등급을 평가하는 기준이 되기도 한다[1,2]. 접근제어 모델은 각 주체와 각 객체의 관계에 따라 접근을 통제하는 구조이다. 접근제어 모델을 적용할 경우, 불법적인 사용자의 시스템으로의 접근이 통제되어 시스템 자원을 안전하게 보호할 수 있지만, 접근제어 모델을 우회해서 접근할 수 있는 알려지지 않는 보안 취약점을 통해 접근이 가능해 질 수도 있다. 백업 등으로 시스템 데이터를 저장하고 있던 디스크가 도난되었을 경우 데이터의 안전성은 보장할 수 없다. 백업이 이루어질 경우 백업을 수행하는 사용자는 시스템의 모든 데이터를 접근할 수 있으므로 데이터의 유출에 대해 안전을 보장할 수 없다.

데이터 암호화는 접근제어 모델이 가지는 이런 문제를 해결해 줄 수 있다. 지금까지 몇몇 대학과 연구기관에서 데이터를 암호화 하는 파일시스템들이 연구되었다[3,4,5]. 하지만, 대부분이 각 사용자 단위의 암호화를 구현하였기 때문에 각 사용자 단위의 키를 관리하기 위한 구조가 필요하였고, 다른 사용자와의 데이터 공유 문제도 발생하였다.

본 논문에서는 암호화 파일시스템에 접근제어 모델을 적용함으로써 기존의 암호화 파일시스템이 안고 있는 구조적 복잡성과 데이터 공유의 문제를 해결하는 구조를 제시한다.

2. 관련 연구

2.1 암호화 파일시스템

데이터를 암호화하는 방법은 여러 가지가 있다. 사용자가 직접 데이터를 암호화하는 방법은 사용 편이성이 떨어지며, 사용자가 복호화한 평문 형태의 데이터를 시스템에 그대로 남겨둘 가능성이 있다. 응용프로그램 수준에서의 데이터 암호화는 사용자에게 편리성을 제공하지만, 각 응용프로그램에서 사용하는 데이터 및 암호화 알고리즘의 호환성 문제가 발생한다[3].

따라서 데이터 암호화는 파일시스템 수준에서 자동으로 이루어지는 것이 사용 및 관리 측면에서 유리하다. 파일시스템 수준에서의 암호화는 사용자로 하여금 데이터 암호화를 위한 특별한 조작을 필요 없게 하는 사용상의 투명성을 제공하고, 시스템 침입자에 대해서 그들이 원하는 정보를 감춤으로써 데이터에 대한 보안성을 제공해 준다. 대표적인 암호화 파일시스템으로는 다음과 같은 것들이 있다.

Cryptographic File System(CFS)는 NFS 파일시스템 내에 암호화 기능을 추가한 것으로, 암호화된 파일에 대해 표준 유닉스 파일시스템 인터페이스를 적용함으로써 시스템 수준에서의 보안 저장장치(secure storage)를 제공한다[3]. 사용자 수준의 구현이기 때문에 많은 문맥 교환 등으로 시스템 성능에 한계가 있다.

Transparent CFS 는 원격 NFS 서버와 통신하는 수정된 클라이언트 쪽의 NFS 커널 모듈이다[4]. 커널 영역에서 구현되어 암호화 서비스와 파일시스템 사이에 더 깊은 통합을 제공함으로써 CFS 를 개선하였다. 사용자 키는 /etc/tcfspasswd 파일에 저장되는 데, 이는 차후에 보안성을 감소시킨다.

Cryptfs 는 Stackable Vnode Layer loadable kernel module 로써 설계 및 구현된 파일 시스템이다[5]. 사용자에게 클라이언트 파일 시스템을 '캡슐화' 함으로써 투명한 암호화 기능을 수행하며 커널 수준에서 동작한다. 암호화 키는 사용자 ID 뿐만 아니라 프로세스 세션 ID 에 기반하고, 커널 메모리에 보관하므로 좀 더 강한 보안성을 제공하지만, 다른 사용자들간의 공유 문제가 어렵다는 단점이 있다.

본 논문에서는 대부분 암호화 파일시스템의 문제로 제시되는 파일 공유 문제를 해결하고, 사용자 별로 유지할 필요가 없는 접근제어를 이용한 간단한 키 관리 구조를 제공한다.

2.2 접근제어모델

본 논문에서 적용할 접근제어 모델은 강제적 접근제어(Mandatory Access Control, MAC)과 역할 기반 접근제어(Role Based Access Control, RBAC)이다. MAC 은 자율적 접근제어(Discretionary Access Control, DAC)에서 발생할 수 있는 문제 즉, 슈퍼 유저일 경우 DAC 를 거치지 않는 문제를 해결할 수 있는 모델이다[1,2]. MAC 은 사용되는 주체와 객체의 등급과 범주로 접근을 검사한다. 범주는 본 논문에서 무시한다. 주체의 등급이 객체의 등급보다 높은 경우 하위 등급의 객체에 대한 읽기가 가능하지만, 주체의 등급과 객체의 등급이 같은 경우에 대해만 쓰기가 가능하다. 이는 토로이안 목마 같은 침입을 방지할 수 있다[1]. RBAC 은 역할에 기반한 접근제어 모델로써, 슈퍼 유저의 권한을 제한할 수 있다. 이럴 경우 슈퍼 유저에게 모든 권한을 주었을 경우 발생할 수 있는 문제들을 해결할 수 있다. 더 나아가서 정해진 역할을 가진 사용자 만이 접근할 수 있는 시스템 자원을 정하여 시스템의 부분별한 접근을 막을 수 있다[1].

3. 설계

제안된 암호화 파일시스템은 손쉬운 파일의 공유 및 간단해진 키 관리 구조를 제공하기 위해 보안 커널의 일부로 접근제어 모듈과 연동되어 동작하도록 설계된다. 암호화 파일시스템은 유닉스 계열의 운영체제 중 하나인 FreeBSD 운영체제를 기반으로 한다. FreeBSD 는 소스가 공개되어 있고, 파일시스템의 개발이 용이한 Stackable file system 을 지원한다. Stackable file system 은 여러 개의 파일시스템을 층으로 쌓아서 연동시킬 수 있다.

그림 1 은 제안된 암호화 파일시스템과 접근제어 모듈과의 연관성을 보여준다.

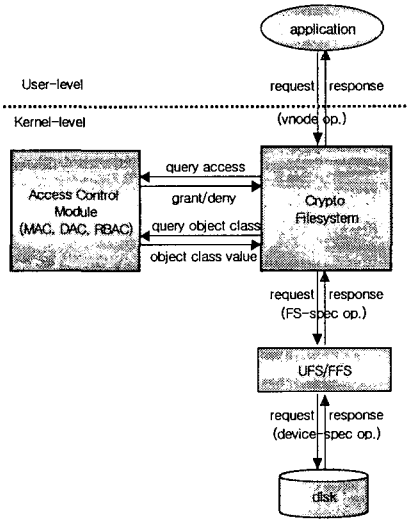


그림 1. 접근제어 모듈을 적용한 암호화 파일시스템

암호화 파일시스템은 [5]에서 소개된 Stcking 기법을 사용하여 커널 내의 vnode 수준에서 설계된다. 이 기법을 이용할 때 얻을 수 있는 이점으로는 기존 파일시스템의 수정 없이 파일시스템에 새로운 기능을 추가할 수 있다. 암호화 파일시스템은 응용프로그램의 요청을 받아 암호화를 수행한 후 하위 레벨의 파일시스템으로 전달하는 역할을 한다.

암호화 파일시스템 내에는 암호화 결정자(Encryption Decider)가 있어, 응용프로그램의 요청으로 데이터를 처리할 때 암호화를 실행할지를 결정한다. 암호화 결정자는 암호화의 기준이 되는 객체의 등급을 얻기 위한 접근제어 모듈과의 인터페이스를 가진

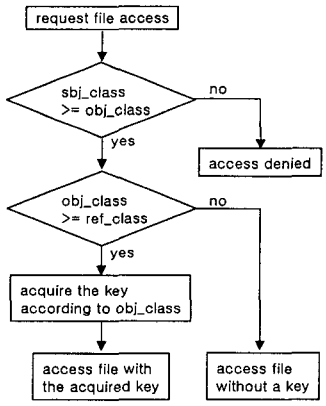
다. 그리고 암호화가 필요할 경우, 해당 객체의 암호화에 필요한 키를 획득하는 역할도 한다. 암호화 결정자에서 암호화를 할 필요가 없다고 결정이 되면, 응용프로그램의 요청은 암호화 부분을 거치지 않고 바로 특정 파일시스템에게 전달된다.

3.1 키관리

기존 암호화 파일시스템에서는 데이터가 사용자 단위의 키로 암호화되었기 때문에 사용자들간의 데이터 공유가 거의 불가능하였다. 접근제어 모델 중 강제적 접근제어(Mandatory Access Control, MAC)는 이러한 한계를 극복할 수 있는 좋은 구조이다[2]. MAC 에서 사용되는 등급 단위로 키를 생성하고, 그 키를 이용하여 파일을 암호화한다면, 같은 등급을 가진 사용자들은 같은 키를 가지므로 파일을 공유할 수가 있다. 따라서 시스템에서는 사용자 별 키에 대한 정보를 유지할 필요가 없고, 사용자가 시스템에서 어떤 등급을 가지는지만 알면 된다. 게다가 시스템에서 암호화하여 저장할 파일의 최소 등급을 정해 놓아 낮은 등급의 파일의 암호화 과정을 생략하여 시스템의 성능 개선시킬 수 있다. 키에 대한 정보는 파일이나 스마트 카드에 저장하고, 접근제어 모듈을 이용하여 보안 관리자만이 접근할 수 있도록 한다. 키 파일의 접근은 시스템이 처음 시작될 경우에만 필요하다.

낮은 등급의 사용자가 높은 등급의 파일을 읽을 수 없는 것은 당연하지만, 높은 등급의 사용자가 낮은 등급의 파일을 읽는 것은 가능해야 한다. 따라서 접근하는 파일을 읽기 전에 액세스하는 파일의 등급에 해당하는 키를 얻어야 한다. 시스템 동작 중에는 시스템에서 사용하는 등급에 해당하는 키들이 커널 메모리에 적재되어 있으므로, 해당 등급의 키를 얻어오면 된다.

다음 그림 2 는 파일을 읽을 경우 키를 얻는 과정을 보여준다.



sbj_class : class of active units (processes or users)
obj_class : class of protectied units (files etc)
ref_class : minimum class to be encrypted

[그림 2] 파일 읽기에서의 키 획득 과정

파일의 읽기는 주체보다 하위 등급인 파일의 액세스가 가능하지만, 파일을 쓸 경우는 보안상의 문제로 주체와 파일의 등급이 같아야 액세스가 가능하다. 따라서 쓰기의 경우의 키 획득 과정은 읽기에서의 키 획득 과정에서 주체와 객체의 등급 비교시 같은 경우에만 통과하도록 해 준다.

3.2 암호 알고리즘과 모드

데이터를 안전하게 저장하기 위해서는 강한 암호화 알고리즘이 필요하다. 현재까지 알려진 자유롭게 사용할 수 있는 블록 알고리즘들 중 하나인 Blowfish 는 비교적 빠르며, 간단한 알고리즘이다[6]. 데이터를 암호화한 후의 크기가 이전과 같기 때문에 파일을 접근하는 기준인 offset 의 수정없이 그대로 사용할 수 있다. 또한 키 길이를 가변적으로 사용할 수 있다는 장점이 있다.

블록 알고리즘은 고정된 크기의 64 비트 데이터를 입출력으로 한다. 따라서 64 비트를 초과하는 평문을 처리하기 위하여 64 비트 단위로 변환하는 과정이 필요하다. 일반적으로 많이 사용되는 방법으로는 ECB(Electronic CodeBook), CBC(Cipher-Block Chaining), CFB(Cipher FeedBack), OFB(Output FeedBack)의 네 가지 모드가 있다[6]. 여기서는 구조적인 취약점을 감소하면서 비교적 간단한 CBC 모드의 암호화가 이루어 지도록 설계한다.

하지만, 전체 파일을 CBC 모드로 암호화할 때, 파일의 크기가 크고 뒷부분의 데이터를 필요로 한다면 파일의 처음부터 복호화를 해야 한다. 이는 성능 상에 있어서 심각한 문제를 유발한다. 따라서 시스템에서 사용하는 가상 메모리 페이지 단위인 4KB 혹은 8KB 로 CBC 모드를 적용시킨다.

3.3 파일의 읽기 및 쓰기

파일을 읽거나 쓰기 위해 파일에 접근하기 전에 시스템에서는 접근제어 모듈을 통해 접근을 시도하는 주체 즉, 프로세스가 접근하려는 파일에 대해 읽기 혹은 쓰기 권한이 있는지를 검사한다. 접근제어 모듈에서 주체의 객체에 대한 접근 권한을 허가할 경우 파일에 대해 읽기 및 쓰기 연산이 수행된다.

응용프로그램이 파일의 읽기를 요청하면 암호화 파일시스템의 암호화 결정자 부분은 파일의 등급과 사용자 등급을 통해 파일을 복호화할 지를 결정하고, 복호화할 필요가 없다면 바로 특정 파일시스템으로 요청을 전달하고, 읽은 데이터를 응용프로그램으로 전달한다. 복호화해야 한다면, 커널 메모리로부터 복호화 키를 얻는다. 복호화를 하기 전에 일단 복호화할 데이터를 특정 파일시스템으로부터 읽어온 다음, 획득한 키를 이용하여 암호화된 데이터를 복호화하여 평문 형태의 데이터를 응용프로그램에게 전달한다.

파일의 쓰기 과정은 읽기 과정보다 복잡하다. 암호화가 페이지 단위로 CBC 모드에 의해 이루어져 있으므로, 수정될 데이터의 경계 부분이 속한 페이지의 내의 수정될 데이터가 속하지 않는 부분은 보존해야 한다. 따라서 일단 수정될 데이터의 경계 부분이 속한

페이지를 읽어와서 수정하지 않을 부분은 보존해 두어야 한다. 따라서 데이터의 복호화도 필요하다.

응용프로그램이 쓰기를 요청하면, 암호화 결정자 부분은 파일의 등급과 사용자 등급을 통해 파일을 암호화할 지 결정하고, 암호화할 필요가 없다면 바로 특정 파일시스템으로 파일 쓰기 요청을 전달한다. 암호화해야 한다면, 객체 등급을 기반으로 암호화/복호화 키를 얻는다. 복호화 과정에서 수정될 데이터의 경계 부분에 속한 페이지를 읽어와 복호화하여 수정되지 않는, 즉 보존해야 할 페이지의 부분을 보존한다. 암호화 과정에서 수정될 데이터를 복호화 과정에서 보존하고 있는 부분과 합쳐 완전한 페이지를 형성한다. 지금까지는 암호화되기 전의 평문 데이터이다. 이제 암호화키를 이용하여 형성된 페이지를 암호화하고, 데이터를 저장하기 위해 특정 파일시스템으로 암호화된 데이터를 전달한다.

4. 보안성

4.1 키파일

암호화에 사용될 키는 시스템 차원에서 관리하며 시스템이 부팅될 때 커널 메모리로 적재된다. 키 파일은 보안 관리자 이외의 사용자가 접근할 수 없도록 하기 위해서, 접근제어 모듈을 이용하여 키 파일의 속성을 설정해 준다. 즉, 접근할 수 있는 사용자의 역할을 보안 관리자로 설정하여 보안 관리자 이외의 사용자는 이 파일에 접근할 수 없도록 한다.

그리고 더 강한 보안성을 제공하기 위해 스마트 카드에 암호화 키를 보관하는 방법이 있다. 시스템이 시작될 경우에만 등급 별 키 값을 읽으면 되고, 그 이후에는 커널 메모리에 적재된 값을 이용한다. 곧 시스템의 저장매체에는 키 값이 남아 있지 않게 된다.

4.2 백업

접근제어 모듈로 인해 불법적인 접근을 통제할 수 있으므로, 시스템 운용 중에는 암호화 파일시스템의 역할이 두드러지지 않는다. 하지만, 데이터를 백업할 경우 암호화된 데이터를 백업할 수 있고, 백업된 매체의 도난에 대해서 데이터의 안전을 유지할 수 있다.

백업 관리자의 경우 사용자의 모든 데이터를 접근할 수 있어야 하므로, 백업 관리자에 의한 고의적인 데이터 유출이 가능해진다. 하지만, 백업 관리자에 대해 암호화 과정을 수행하지 않고 지나치게 한다면, 유출되는 데이터는 암호화된 형태이므로, 데이터의 안전성을 보장할 수 있다.

또한 접근제어 모듈을 우회하여 데이터에 접근할 경우 정상적인 파일의 읽기 과정이 수행되지 않으므로 즉, 정상적인 키 획득 과정이 생략되므로, 접근제어 모듈을 통과한 불법 침입자에게는 데이터를 노출시키지 않는다.

5. 결론 및 향후과제

암호화 파일시스템은 데이터의 암호화 저장이라는 방법으로 사용자의 데이터를 보호하는 기능을 제공해 준다. 하지만, 주로 사용자 단위로 암호화가 이루어지기 때문에 사용자들간의 데이터 공유 문제를 어렵게 하였다. 암호화 키의 생성은 사용자 기반으로 이루어지고, 생성된 키는 사용자 별로 관리를 해주어야 하는 단점도 있다.

본 논문에서는 접근제어 모델을 적용시켜 시스템 운용에 큰 과부하를 발생시키지 않으면서, 사용자들간의 손쉬운 데이터 공유와 간단한 키 관리를 제공하는 암호화 파일시스템의 구조를 제안하였다. 또한 접근제어 모델만을 적용하였을 경우 발생할 수 있는 문제 즉, 백업 관리자의 사용자 데이터 열람 가능, 백업 매체 도난 및 접근제어를 우회한 데이터 접근 등을 암호화 파일시스템으로 해결할 수 있을 것이다. 데이터의 암호화는 시스템의 물리적인 보안을 이룰 수 있다는 점에서 꼭 필요한 방법이다.

향후에는 제안된 시스템의 구현을 통해 이미 구현된 기존 암호화 파일시스템과의 성능 비교와 키 파일의 보관을 시스템 내에서가 아니라 스마트 카드 같은 매체의 도입으로 시스템과 독립시켜 보안성을 강화하는 구조의 연구가 필요하다.

파일의 보안 등급이 변경될 경우 변경된 보안 등급에 해당하는 키로 파일을 다시 암호화하는 방법이 요구된다. 이 때 시스템에서 파일의 보안 등급 변경을 인식하여 자동으로 처리할 수 있도록 하는 방법도 연구되어야 할 과제이다.

참고문헌

[1] J. G. Ko, S. Y. Doo, S. K. Un and J. N. Kim. "Design and Implementation for Secure OS based on Linux," Proceedings of WISA2000, Vol I. pp.175-181, November, 2000

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

[3] M. Blaze. "A Cryptographic File System for Unix," Proceedings of the first ACM Conference on Computer and Communications Security (Fairfax, VA). ACM, November, 1993

[4] G. Cattaneo and G. Persiano. "Design and Implementation of a Transparent Cryptographic File System for Unix," Unpublished Technical Report. Dip. Informatica ed Appl, Universita di Salerno, 8 July 1997

[5] E. Zadok, I. Badulescu, and A. Shender. "Cryptfs: A Stackable Vnode Level Encryption File System," Technical Report CUCS-021-98. Computer Science Department, Columbia University, 28 July 1998

[6] B. Schneier. In Applied Cryptography, Second Edition. John Wiley & Sons, 1996