Linux 운영체제 동적 모듈 개념을 이용한 보안 파일 시스템 모듈 설계

장 승 주† · 이 정 배††

요 약

본 논문은 Linux 운영체제에서 동적 모듈 개념을 이용하여 보안 파일 시스템을 설계하였다. Linux 운영체제에서 동적 모듈 개념을 이용할 경우 커널 소스를 수정하지 않고 사용자가 원할 경우 커널 모듈을 언제든지 추가할 수 있는 장점을 가진다. 보안 파일 시스템은 사용자 데이터를 암호화함으로서 접근이 허용되지 않은 사용자에게 중요한 자료가 노출되지 않도록 해준다. 보안 파일 시스템은 blowfish 알고리즘을 사용하여 암호 및 복호화를 수행한다. 키 생성을 위하여 키 서버를 통하지 않고 자체 키 생성 알고리즘에 의하여 키를 만든 후 이 키 값을 이용한다.

Design of the Security Cryptography File System Based on the Dynamic Linking Module on the Linux O.S

Seung-Ju Jang† · Jeong-Bae Lee††

ABSTRACT

We propose the Security Cryptography File System to encrypt or decrypt a plaintext or an encrypted text by using the dynamic linking mechanism in the Linux kernel. The dynamic linking mechanism gives the flexibility of the kernel without changing the kernel. The Security Cryptography File System uses the blowfish algorithm to encrypt or decrypt a data. To overcome the overhead of the key server, I use key generating algorithm which is installed in the same Security Cryptography File System. The Security Cryptography File System is fitted into the Linux system.

키워드 : 보안 파일 시스템(Security Cryptography File System), 동적 연결 모듈(Dynamic Linking Module), 데이타 암호/복호, 키생성 (Encryption/Decryption of a Data, Key Generation)

1. 서 론

동적 모듈은 Linux 운영체제에서 제공하는 개념으로, 사용자가 커널의 소스 코드를 변경하지 않고서 자신이 원하는 커널의 기능을 시스템이 동작 중에 적재 또는 삭제가 가능하도록 하는 기능이다. Linux의 동적 모듈 기능은 커널 내에 필요한 기능을 필요한 시점에 적재가 가능하도록 해준다.

일반적인 파일 시스템 보안은 데이터 암호화, 프로세스 실행 제어, 접근 권한 제어 등을 포함한다. 유닉스 시스템은 다수의 사용자들이 동시에 사용할 수 있기 때문에 각 사용자의 데이터를 보호하기 위한 기능이 필요하다. 이를 위해 유닉스 파일 시스템에서는 파일이나 디렉토리에 사용자들에 대한 여러 가지 권한을 지정할 수 있다. 그리고 실행 파일인 경우 그 파일이 실행될 때 어떤 사용자의 권한으로 실행이 되어야 하는지도 지정할 수 있다. 이와 같이 사용자의 권한을 명확히 하고, 사용상의 불편이 없도록 여러 가지 기능을 제공하는데, 이 중 몇 가지는 사용상의 편의성을 제공하는 반면 보안상의 허점을 지니고 있어 공격의 대상이 되기도 한다. 이와 같이 보안 기능만을 강화했을 경우는 사용자에게 사용상의 불편함을 강요해야하고, 반면 사용상의 편의성 만을 고려했을 경우는 외부 공격에 취약함을 드러내게 된다. 따라서 사용상의 편의성과 안전한 보안 기능을 동시에 수용할 수 있는 방안을 연구해야 할 것이다[2, 3].

사용자의 데이터를 안전하게 보호하기 위해서 암호화/복호화 할 때 고려되어야 할 사항으로는 파일 데이터, 파일

※ 본 논문은 (주)나일 소프트의 연구비 지원을 받아 이루어졌음.
본 논문은 2003년도 동의대학교 학술연구 조성비에 의해 연구되었음.
† 정 회 원 : 동의대학교 컴퓨터공학과 교수
†† 종신회원 : 선문대학교 컴퓨터정보학부 교수
논문접수 : 2003년 7월 1일, 심사완료 : 2003년 9월 22일

명, 파일 속성 등이 있다. 파일 데이터는 저장된 내용을 다른 사용자 혹은 침입자에게 노출시키지 않도록 암호화하여 저장한다. 파일명은 파일의 이름을 통하여 이 파일이 어떠한 내용을 가지고 있는지 추측할 수 있는 가능성에 대비하여 구현된 기능이다. 파일 속성은 파일에 대한 소유권과 모드에 관련된 사항이다. 같은 그룹에 속한 사용자가 암호화된 파일에 접근 할 경우, 사용자마다 다른 암호화 키를 사용하더라도 파일에 접근할 수 있도록 해야하고, setuid bit 설정을 방지하는 기능도 포함하고 있다.

본 논문에서는 외부의 접속을 파악하고 운영체제의 파일 시스템 레벨에서 보안 기능을 제공할 수 있는 커널 모듈을 구현한다. 네트워크를 통한 통제를 뚫고 침입해오더라도 데이터가 저장된 파일 시스템에 접근 할 수 없다면 내부의 중요한 자료 유출은 방지할 수 있을 것이다.

본 논문에서 구현한 보안 파일 시스템 모듈은 Linux의 동적 적재 모듈을 사용하여 기존의 커널 소스에 대한 변경 없이 커널 레벨에서 보안 기능을 제공할 수 있도록 구현하였다.

현재 Unix 기반 뿐 만 아니라 Windows 계열에서 사용되는 보안 파일 시스템이 다수 개발되어 있기는 하지만 사용상의 불편함 때문에 널리 사용되지 못하고 있다. 기존의 보안 파일 시스템은 커널 소스를 변경하고, 사용자의 설정을 요구하기 때문에 보안 기능의 장점을 살리지 못하고 있다. 본 논문의 구성은 2장에서 관련 연구에 대하여 살펴보고, 3장에서는 보안 파일 시스템 모듈의 설계를, 4장에서 결론을 내리고자 한다.

2. 관련 연구

보안 파일시스템은 1993년 AT&T Bell 연구소의 Blaze에 의해서 CFS(Cryptographic File System)로 제안되었다[2, 3, 8, 9]. 기존에 여러 가지의 보안 도구들이 존재했지만 운영체제 레벨에서 암호화 기능을 제공하는 시도는 근래에 들어서 이루어졌다. 이후 CFS의 기능을 보완한 TCFS(Transparent Cryptographic File System) 등의 보안 파일 시스템들이 개발되었다[2-6, 14, 15]. 현재 개발되었거나 제안된 보안 파일 시스템들은 크게 두 가지의 형태로 구성되어 있다. 첫 번째는 커널에 포함된 형태이고 다른 형태로는 User-level File System이다. 커널에 포함된 파일 시스템의 경우는 개발이나 디버깅이 어렵다[4]. 이러한 형태의 파일 시스템을 개발 할 경우에는 low level의 디바이스와 운영체제에 대한 기능을 충분히 이해하고 있어야 하지만 커널에 상주하기 때문에 성능 면에서는 우수하다. 반면, User-level에 구현된 파일 시스템은 구현이 용이한 반면 성능 면에서는 뒤떨어진다.

최근에는 이러한 장·단점을 보완한 형태의 파일 시스템이 개발되고 있는데, VFS(Virtual File System) 계층을 사용한 모듈화된 형태의 파일 시스템이다. 이러한 파일 시스템은 광범위하게 사용되고 있는 VFS layer상에 추가의 계층을 삽입하여 기존의 커널 소스의 변경을 최소화하고 개발의 용이성과 커널 코드와 같이 동작하여 성능 면에서도 우수한 면을 보여주고 있다. 새롭게 추가된 모듈 형태의 파일 시스템에서는 VFS와 기존 파일 시스템 사이의 정보들을 가로채어 암호화/복호화 기능을 수행한다[1, 5-9].

또한 지역적인 관점에서 분류를 한다면 local file system의 보안을 지원하는 파일 시스템과 NFS(Network File System)[10, 11] 기반의 remote file system의 보안을 지원하는 것으로 분류할 수 있다. 대부분의 보안 파일시스템이 Unix/Linux 운영체제 기반에서 사용되며, EFS(Encrypting File System)는 최근에 발표된 Windows 2000에서 사용되는 파일 시스템인 NTFS에 보안 기능이 추가된 것이다[1, 10, 14]. 이들의 구현방식을 분류하면 <표 1>과 같다.

〈표 1〉 보안 파일 시스템 구현 분류

	NFS 기반	Local 기반
User-Level	CFS	SFS
Kernel-Level	TCFS	CryptFs EFS

<표 1>에서 CFS(Cryptgraphic File System)와 TCFS(Transparent Cryptographic File System)는 NFS기반의 네트워크 파일 시스템으로 구성되어 있으며, SFS(Secure File System), CryptFS, EFS는 지역 파일 시스템의 보안에 기반을 두고 있다[11-13]. 파일시스템이 존재하는 영역에 따라 분류하면 TCFS, CryptFS, EFS는 커널 영역에 존재하며 CFS, SFS는 사용자 영역에서 데몬 형태로 동작한다.

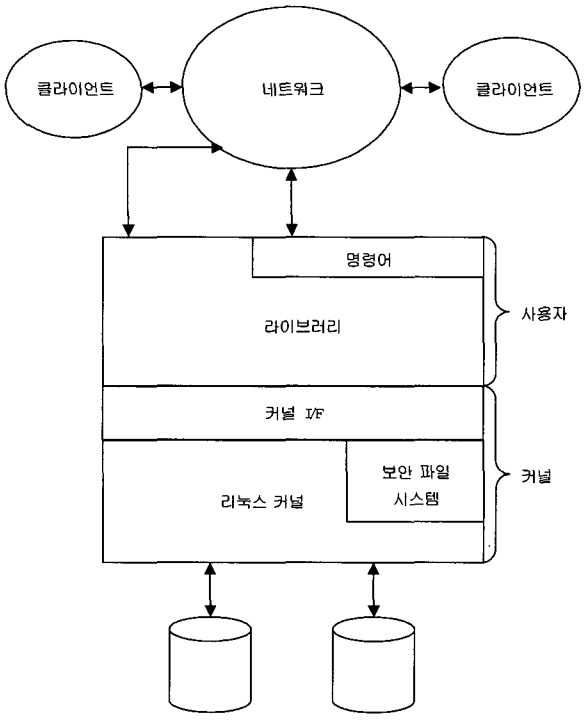
3. 동적 보안 파일 시스템 설계

3.1 설계 환경

보안 파일 시스템 설계 환경은 Linux 운영체제 상에서 커널 버전은 2.2.X 버전을 사용하였다. 또한 컴파일러는 GNU의 gcc를 이용하였다. 보안 파일 시스템의 전체적인 구조는 (그림 1)과 같다.

3.2. 설계 내용

보안 모듈은 최대한 사용자의 편의성을 고려하고 사용상의 투명성(transparency)을 제공할 수 있도록 설계하였다. 보안 모듈은 크게 두 가지의 기능을 가진다. 사용자의 데이



(그림 1) 보안 파일 시스템 구조

터를 암호화하여 저장하는 기능과 특정 사용자에 대한 프로세스 실행 통제 기능이다.

암호화는 개별적인 파일이 아닌 디렉토리에 포함된 내용을 암호화하는 방식을 사용하며, 키는 사용자별로 고유의 키를 할당하여 사용한다. 사용자는 암호화된 파일을 저장하기 위한 디렉토리를 생성한 후 그 디렉토리에서 일반적인 파일을 다루는 것과 동일하게 작업을 수행하면 된다. 현재 구현된 모듈에서는 각 사용자의 홈 디렉토리에서 암호화하여 저장할 디렉토리를 생성하고, 모듈의 설정 파일에 디렉토리를 등록하면 이 디렉토리 안에 포함된 파일과 서브디렉토리의 내용들이 암호화되어 저장되고, 파일에 대한 작업 시 복호화 되도록 설계하였다. 암호화에 관련된 세부사항은 모듈에서 모든 것을 처리하기 때문에 사용자는 생성된 디렉토리를 등록하는 것 외에 다른 설정을 하지 않아도 되고, 일반적인 파일 연산과 동일한 작업을 수행하면 된다.

파일의 암호화를 위해 사용되는 알고리즘은 대칭형 알고리즘인 Blowfish를 사용한다. Blowfish 알고리즘은 DES(Data Encryption Algorithm)에 비해서 속도가 빠르고 안전하며, 다양한 언어에서 사용할 수 있도록 구현되어 있다. 암호화와 복호화에 사용되는 키는 동일하며 128bit의 키를 사용하고 블럭의 크기는 64bit이다.

디렉토리에 대하여 암호화 여부를 결정한 이유는 파일 단위의 암호화를 수행할 경우에는 파일에 대한 추가 속성 등을 정의하여야 하며 따라서 기존 커널의 소스 코드를 변경해야하기 때문이다. 구현한 보안 모듈은 기존의 커널 소스는 전혀 변경하지 않고 암호화 기능을 사용할 수 있도록 설계한다. 그 외에 SFS(Secure File System)와 같이 암호화된 파일의 목록을 가지도록 설계할 경우 빈번히 발생하는 파일 접근 요구에 대하여 목록을 매번 검사해야하는 추가의 오버헤드가 발생하므로 성능상의 문제점을 가진다.

선언된 시스템 호출 테이블을 사용하여 기존의 시스템 호출을 변경하는 방법은 (그림 2)와 같다. 변경 작업은 "init_module"에서 이루어지며, 원래의 상태로 돌려놓는 것은 "cleanup_module"에서 이루어진다.

```
int init_module(void)
{
        printk ("<1>Initialize Crypt Module !!!\n") ;
        org_write = sys_call_table [ __NR_write] ;
        org_read = sys_call_table [ __NR_read] ;
        org_ioctl = sys_call_table [ __NR_ioctl] ;

        sys_call_table [ __NR_write] = encrypt_write ;
        sys_call_table [ __NR_read] = decrypt_read ;
        sys_call_table [ __NR_ioctl] = new_ioctl ;

        return 0 ;
}
```

(그림 2) "init_module"에서 시스템 호출 변경

(그림 2)에서의 예는 3개의 시스템 호출 함수를 변경하는 것이다. 선언된 시스템 호출 테이블의 상수 값을 이용해서 write, read, ioctl 함수의 주소를 저장한다. 그리고 모듈에서 새롭게 정의한 함수를 시스템 호출 테이블의 엔트리에 새롭게 등록한다. "init_module"에서 이러한 과정을 거치면 간단하게 기존의 시스템 호출을 변경할 수 있다.

```
void cleanup_module ()
{
      printk ("Unload Crypt Module !!!\n") ;

      sys_call_table [ __NR_write] = org_write ;
      sys_call_table [ __NR_read] = org_read ;
      sys_call_table [ __NR_ioctl] = org_ioctl ;

      if (new_user_key != NULL)
            kfree (new_user_key) ;
}
```

(그림 3) "cleanup_module"에서 시스템 호출 복원

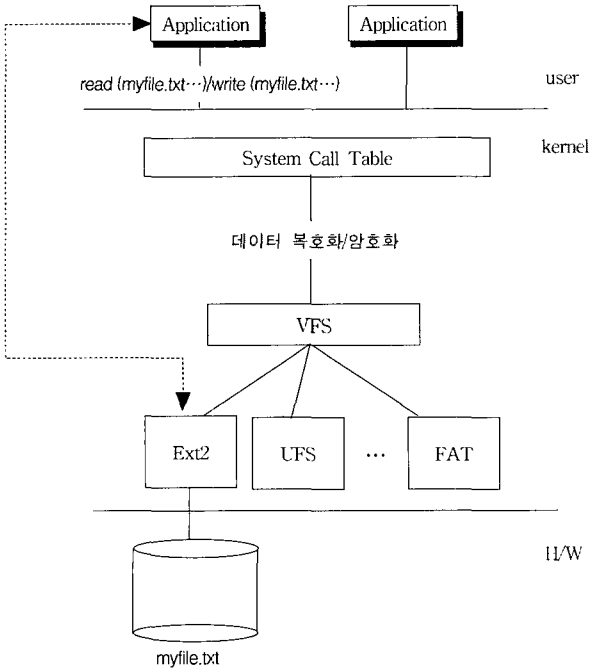
(그림 3)의 예는 모듈이 커널에서 삭제되기 전에 수행되어지는 "cleanup_module"의 코드이다. "cleanup_module"에서는 초기화 작업의 반대 작업으로 사용하던 자원을 해제하는 기능을 수행한다. "init_module"에서 변경한 시스템 호

출 함수를 원래의 주소로 복원하고 모듈이 사용한 메모리 등은 완전히 반환해야한다. "cleanup_module"에서 이러한 작업을 수행하지 않으면 시스템에 치명적인 문제가 발생할 수 있기 때문에 반드시 처리해야하는 부분이다.

모듈에서 데이터를 암호화하거나 복호화하는 핵심 부분은 write, read 함수 부분이다. write에서는 디스크에 저장하려는 데이터를 암호화하는 기능을 수행하고, read에서는 디스크에 저장된 데이터를 읽어들여서 복호화하고 사용자에게 리턴하는 기능을 수행한다.

3.2.1 보안 파일 시스템 파일 접근 관련 커널 인터페이스

개발된 보안 파일 시스템에서 보안 모듈은 VFS(Virtual File System) 계층의 상위에서 암호화 과정을 거친 후 데이터를 VFS로 전달하는 방식이다. 즉, 커널로 전달되는 시스템 호출을 필터링하여 파일에 관련된 호출일 경우에는 데이터를 암호화 혹은 복호화하는 것이다. 먼저 일반적인 시스템 호출이 발생하여 파일에 관련된 작업을 수행하는 과정을 살펴보면 (그림 4)와 같다.



(그림 4) 보안파일시스템에서 read · write operation 과정

응용 프로그램에서 발생한 read 요청은 system call table을 거쳐서 VFS로 전달되고 VFS는 이 요청이 어떠한 파일 시스템에 대한 요청인지를 확인하고 요청의 대상이 되는 파일 시스템으로 이 read 요청을 전달하게 되고 응용 프로그램에서는 이에 대한 응답을 받게된다[1, 11, 12].

개발된 보안 파일 시스템 모듈은 개발과 사용상의 용이성을 제공할 수 있는 동적 모듈 방식을 사용하였다. 모듈의 암호화/복호화 기능은 system call table의 관련 함수들을 변경하여 새롭게 작성된 함수들이 호출되어 사용되게 함으로써 이루어진다. 즉, 모듈이 로드되면서 기존의 system call table의 함수를 새롭게 작성한 함수로 변경하고, 응용 프로그램에서 호출이 발생하면 새로운 함수가 VFS로 전달되게 하는 방식이다. 이렇게 할 경우에는 기존 커널의 다른 부분은 전혀 변경하지 않고도 새로운 기능을 커널에 추가할 수 있다.

가장 핵심이 되는 부분은 파일을 읽고, 저장하는 부분이다. 파일을 암호화하고 암호화된 내용을 다시 복호화하여서 읽어들이는 기능을 제공해야 하기 때문에 write 함수에서 파일을 암호화하여 저장한다. 또한 read 함수에서 암호화된 내용을 복호화하여 응용 프로그램으로 전달한다. 이러한 과정의 전반적인 동작은 (그림 4)와 같다. (그림 4)의 예에서는 myfile이라는 파일에 대해서 읽기·쓰기 요청이 발생하여 처리되는 과정을 보여주고 있다.

3.2.2 암·복호화 기본 기능

암호 시스템은 암호화가 적용되는 평문 길이의 최소 단위가 한 개의 비트나 문자이면 스트림 암호(stream cipher), 한 개 이상일 경우에는 블럭 암호(block cipher)로 각각 분류된다. 파일은 블럭 혹은 페이지 단위로 처리되기 때문에 파일 암호화에는 블럭 암호화 기법이 사용된다.

RSA(Ronald Rivest, Adi Shamir and Leonard Adelman)와 같은 비대칭형 알고리즘은 암호화 및 복호화에 소요되는 시간 및 계산량이 비교적 많기 때문에 일반적으로 길이가 긴 메시지를 암호화하는 데에는 사용하지 않는다. 대부분 식별번호와 같은 길이가 짧은 데이터를 암호화하는 데에 이용된다. 특히, 대칭형 암호에 사용되는 대칭키를 암호화하여 분배하는 목적으로 공개키 암호가 사용되고, 대칭키를 이용하여 메시지를 암호화하는 것이 일반적이다[10, 13].

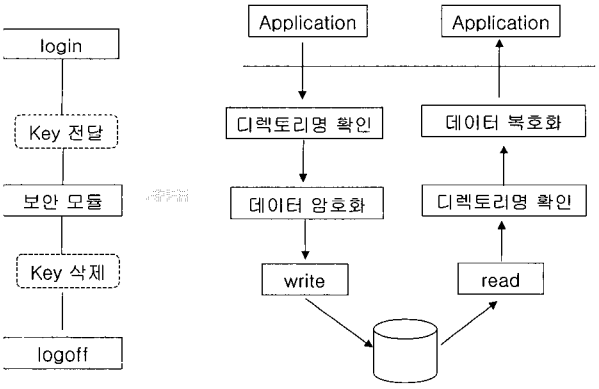
Blowfish는 DES(Data Encryption System)를 대체하기 위한 새로운 대칭키 블럭 암호화 알고리즘으로서 1993년 BRUCE SCHNEIER에 의해 만들어진 알고리즘이다. 블럭 크기는 64비트이며, 키 길이는 32~448비트까지 가변적으로 사용할 수 있다. Blowfish 알고리즘은 2개 부분으로 구성되어 있다. Key 확장부분과 data 암호화 부분이다. 키 확장부분은 최대 448bit의 키를 전체 합쳐서 4168byte를 갖는 여러 개의 subkey로 변환한다. 데이타 암호화는 16round를 통해 발생한다. 각 round는 키에 따른 순서의 교환(permutation) 과 키와 데이타에 따른 대체(substitution)로 구성된다. 모든 작업은 XOR과 32bit 워드로의 가산이다. 유일한 추가 작업은 매 round마다 4개의 인덱스된 배열 데이타를 조사하는 것이다. 이들 키들은 암호화/복호화 전에 미리

계산되어져야 한다. 복호화 과정은 암호화 과정과 동일하며, 데이터 프레임을 역순으로만 사용하면 된다. Blowfish에서 사용되는 블럭 암호화 운영모드는 4가지 방식이 있다[10].

3.2.3 보안 파일 시스템 암호화/복호화 기능

사용자에 대한 정보가 전달된 후에는 이 정보를 바탕으로 모듈에서 암호화를 할 것인지, 복호화를 할 것인지에 대한 결정과 파일에 대한 접근 권한을 검사하게 된다. 먼저 각 사용자는 암호화할 디렉토리를 모듈의 설정파일에 등록해야 한다. 모듈은 이 설정 파일의 내용을 읽어들여 암호화 디렉토리를 판단하게 된다. 파일에 대한 읽기·쓰기 요청이 발생할 경우, 요청의 대상이 되는 파일의 전체 경로명을 얻어내고 설정 파일에 등록된 디렉토리인지를 확인하게 된다. 파일이 등록된 디렉토리 또는 그 하위 디렉토리에 있을 경우에는 암호화·복호화를 수행할 수 있게 된다.

다음으로는 파일에 대한 접근 권한을 검사한다. 이러한 과정을 거친 후, 응용 프로그램에서의 쓰기 요청에 대해서는 데이터를 암호화하여 디스크에 저장하게 되고, 읽기 요청에 대해서는 암호화된 데이터를 복호화하여 사용자에게 전달하게 된다. 이러한 수행 과정을 도식화하면 (그림 5)와 같다.



(그림 5) 보안 파일 시스템 암호화/복호화 모듈의 수행 과정

데이터 암호화는 사용자의 데이터(Tx(i))를 입력으로 받아서 TtoE 알고리즘을 수행하고 나면 암호화된 데이터(Ex(i))가 최종적으로 생성된다. 먼저 사용자로부터 데이터를 읽어들인다(Tx(i), E1 문장). 사용자 데이터를 이용하여 응용 프로그램을 수행한다(E2 문장). 이것이 수행되고 나면 현재 시스템에서의 작업 디렉토리(CURRENT_PATH)를 얻는다(E3 문장). 현재 작업 디렉토리가 보안 파일 시스템으로 설정이 된 경우는 본 논문에서 제안하는 암호화 과정을 수행하게 된다(E4-E14). 최종적으로 암호화된 파일은 디스크에 저장된다(E10 문장). 전체적인 수행과정은 (그림 6)과 같다.

```
Input : Text data Tx (i)
Output : Encrypted data Ex (i)
Algorithm : TtoE
E1 : Get input data Tx (i) from a user
E2 : Tx (i) := App (Tx (i)) ;
E3 : get CURENT_PATH ;
E4 : if (CURRENT_PATH = SCFS) begin
E5 :    if (uid = CFS_uid) begin
E6 :        Key (uid) := read(key_list, uid) ;
E7 :        while (Tx(i) != EOF) begin
E9 :            Ex(i) := Enc (sys_buf (Tx (i)), Key (uid)) ;
E10 :           write (Ex (i), sys_buf) ;
E11 :       end
E12 :   end
E13 : end
```

(그림 6) 보안 파일 시스템에서 암호화 과정

데이터 복호화는 암호화하는 과정의 반대 순서를 거친다. 암호화 시에는 VFS 계층의 상위에서 암호화를 한 후 VFS 계층으로 전달하지만, 복호화 시에는 VFS 계층에서 암호화된 데이터를 받아들이고 key list에서 사용자의 키를 가져온 후 데이터를 복호화하여 사용자에게 전달하는 과정을 거치게 된다. 즉, 기존의 read 함수의 기능을 이용해서 디스크에 저장된 파일의 내용을 읽어들이고 사용자에게 이 내용을 전달하기 전에 버퍼의 내용이 암호화되어 있는지를 확인한 후, 암호화되어 있으면 버퍼의 내용을 복호화하여 반환하게된다. 전체적인 수행과정은 (그림 7)과 같다.

```
Input : Encrypted data Ex (i)
Output : Text data Tx (i)
Algorithm : EtoT
D1 : App (req (read, uid)) ;
D2 : find sys_read ;
D3 : get CURENT_PATH ;
D4 : if (CURRENT_PATH = CFS) begin
D5 :    if (uid = CFS_uid) begin
D6 :        Key (uid) := read (key_list, uid) ;
D7 :        read (sys_buf, size) ;
D8 :        while (Ex (i) != EOF) begin
D9 :            sys_buf (Tx (i)) := Dec (sys_buf (Ex(i)), Key (uid)) ;
D10 :           copy (sys_buf (Tx(i), buf) ;
D11 :       end
D12 :   end
D13 : end
```

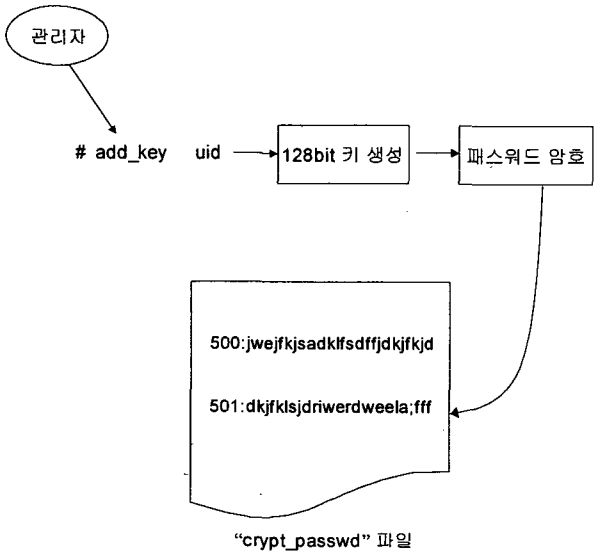
(그림 7) 보안 모듈에서 복호화 과정

3.2.4 키 생성

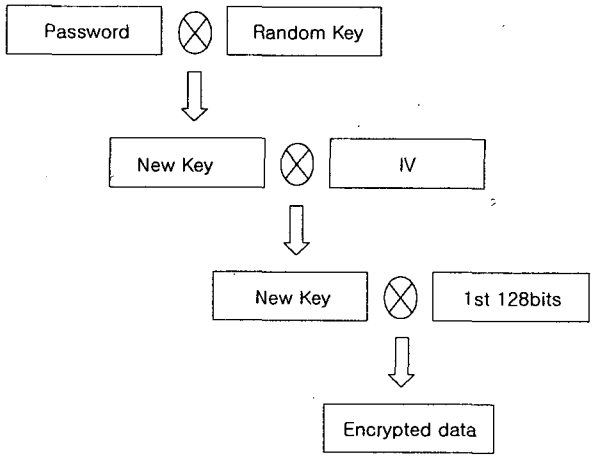
키는 관리자에 의해서 사용자별로 할당한다. 키 생성을 위해서 "addkey"라는 관리용 응용 프로그램을 사용하며, 생성된 키는 "/etc/crypt_passwd" 파일에 저장한다. 이 파일에는 각 사용자에 대한 키와 UID가 함께 저장된다. 키 생성은 random key 생성기를 통하여 임의의 문자열을 생성

하고, MD5(Message Digest 5) 알고리즘을 사용하여 128bit의 길이를 가지는 키를 생성한다. MD5는 가변적인 길이의 메시지를 입력으로 받아들여 128bit의 해쉬 값을 출력하는 해쉬 알고리즘이다.

파일을 암호화/복호화 하는데 사용되는 키가 외부에 노출될 경우에는 파일의 안전성을 보장할 수 없다. 따라서 키를 외부에 노출시키지 않도록 하기 위한 방안으로 키에 대한 암호화 작업을 거친 후 최종적으로 "crypt_passwd" 파일에 저장되게 된다. 키에 대한 암호화는 키를 생성한 후에 사용자의 패스워드로 암호화하는 방식을 사용한다. 사용자별로 암호화에 사용할 키를 생성하는 것만이 관리자가 유일하게 해주어야하는 작업이다.



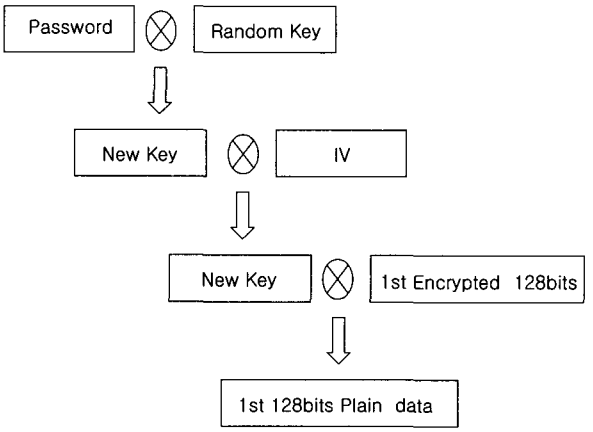
(그림 8) 키 생성과 저장 과정



(그림 9) 키를 이용하여 암호화 하는 과정

새로운 키를 생성하는 과정은 (그림 9), (그림 10)과 같다. 키의 생성 과정은 먼저 사용자가 시스템에 접근할 때 로그인(login)을 하게 된다. 이때 사용하는 패스워드를 입력값으로 이용한다. 이 값을 이용하여 키 값에 대해서 사용권을 가진 사용자 이외에는 접근을 허용하지 않는다. 먼저, 랜덤 키 값을 생성한다. 사용자 패스워드 값 $Passwd_i$과 랜덤하게 만들어진 키 값으로 XOR 연산을 수행한다. XOR한 결과 값을 벡터 값(iv)으로 XOR 연산을 수행한다. 이 값에 대해서 암호화를 하기 위하여 BF 함수를 호출한다.

이미 만들어진 키에 대해서 암호화/복호화를 하는 과정은 (그림 9), (그림 10)과 같다.



(그림 10) 키를 이용하여 복호화하는 과정

4. 실 험

(그림 11)는 "user1"이라는 사용자가 "myfile"이라는 문서를 작성하고 내용을 확인하는 예이다. "secure" 디렉토리에서 작업을 하더라도 일반적인 파일을 다루는 과정과 동일하게 처리된다. 새로운 파일을 작성하게 되면 디스크에는 파일의 데이터가 암호화되어 저장된다. 반대로 (그림 9)에서처럼 "cat" 명령으로 파일의 내용을 확인하면 암호화된 파일 데이터가 복호화되어 정상적인 내용으로 보여지게 된다.

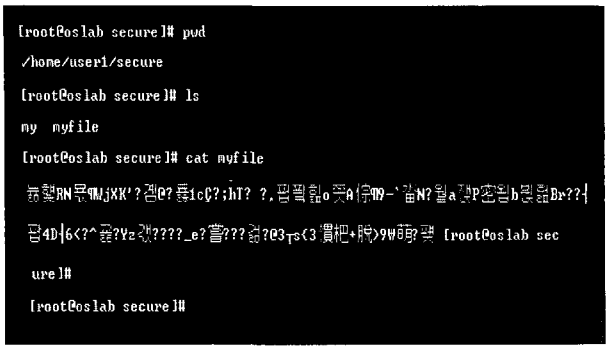
```
[user@oslab secure]$ ls
myfile
[user1@oslab secure]$ cat myfile
Writing kernel code is an arcane art, especially
when a 'real' multi-tasking operation system is
involved, isn't it?

I want to dispel that my and show you that the
life of a Linux kernel programmer is …
```

(그림 11) "secure"디렉토리에서의 파일 작성 실험

이번에는 다른 사용자가 "user1"의 "secure" 디렉토리의 파일에 접근하였을 때 어떤 내용을 보여주는지 확인한다. 실험을 위해서 수퍼 유저가 "user1"의 "secure" 디렉토리에

있는 "myfile"의 내용을 확인한 결과를 보면 다음과 같다.



(그림 12) 암호화된 파일에 대한 외부의 접근

(그림 12)는 수퍼 유저가 암호화 파일로 저장된 "myfile" 파일의 내용을 확인하려 했을 때의 결과이다. 동일한 파일을 수퍼 유저가 확인하더라도 파일의 데이터를 볼 수 없다. 각 사용자가 가지고 있는 암호화 키가 틀리기 때문에 알아볼 수 없는 문자로 복호화되어 보이게 된다.

5. 결 론

Linux 운영체제에서 동적 적재 모듈 메커니즘을 사용하여 커널 레벨에서 파일 암호화/복호화를 지원할 수 있는 보안 파일 시스템을 제안한다. 본 논문에서는 사용자의 데이터를 보호할 수 있는 암호화·복호화 기능을 커널에 추가하였다. 기존의 파일 시스템 자체에도 다양한 접근 권한과 보안 기능이 포함되어 있지만, 외부 침입자에 의해 수퍼 유저의 권한을 얻어서 파일 시스템의 취약점을 가장 많이 이용하고 있다. 보안 파일 시스템 모듈에서는 이러한 문제가 발생하더라도 중요한 정보를 외부의 침입에 대해 보호할 수 있는 기능을 제공한다.

보안 기능은 시스템의 안전성 뿐만 아니라 편의성도 고려되어야 한다. 보안 기능이 너무 엄격하게 유지된다면 사용자들의 불편이 그만큼 커지게 되기 때문이다. 따라서 데이터를 보호할 수 있는 보안 기능 뿐 아니라 기존의 사용자들이 불편함을 겪지 않도록 사용상의 편의를 제공할 수 있도록 설계하였다.

보안 파일 시스템 모듈은 Blowfish 암호화 알고리즘을 사용하여 데이터를 암호화·복호화하게 된다. 암호화에 사용되는 키는 사용자의 패스 워드로 암호화하여 외부의 노출에 대비한다. 보안 모듈은 커널 레벨에서 동작하므로 성능상의 오버헤드도 최소화하였으며, 기존의 커널 소스에 대한 수정 없이 개발되었기 때문에 다른 Linux 시스템으로의 확장도 용이하다.

앞으로의 개선 사항으로는 보다 안전한 키 관리 방안을 적용하는 것과 모듈의 안전성을 보완하는 것이다. 패스워드에 의한 키 보호보다는 보다 안전성이 검증된 키 관리 메커니즘을 연구할 것이다.

참 고 문 헌

[1] Maurice J. Bach, "The Design of The Unix Operating System," Prentice-Hall International Editions, 1987.

[2] M. Blaze, "A cryptographic file system for unix," In First ACM Conference on Communications and Computing Security, pp.158-165, 1993.

[3] Matt Blaze, "Key Management in an Encrypting File system," In Proceedings of the 1994 Summer USENIX Conference, Boston, Massachusetts, USA, June, 1994.

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

[5] Simson Garfinkel, Gene Spafford, "Practical UNIX & Internet Security," O'Reilly & Associates, 1996.

[6] M. K. McKusick, W. N. Joy, S. J. Leffler and R. S. Fabry, "A fast file system for UNIX," ACM Transactions on Computer Systems, 2(3), pp.181-97, Augst, 1984.

[7] Ori Pomerantz, "The Linux Kernel Module Programming Guide," GLP(Linux Document Project) Guide, 1999.

[8] B. Schneier, "Fast Software Encryption," Cambridge Security Workshop Proceedings, (December, 1993), Springer-Verlag, pp.191-204, 1994.

[9] B. Schneier, Blowfish, "Applied Cryptography, Second Edition," John Wiley & Sons, pp.336-3399, 1996.

[10] Douglas R. Stinson, "Cryptographic Theory and Practice," CRC Press, 1995. [8] D. S. H. Rosenthal, "Requirementss for a Stacking Vnode/VFS interface," Unix International document SD-01-02-N014. UNIX Internaional, 1992.

[11] Erez Zadok, Ion Badulescu, Alex Shender, "Cryptfs : A Stackable Vnode Level Encryption File System," Columbia U. CS TechReport CUCS-021-98, 1998.

[12] Erez Zadok, Ion Badulescu, "A Stackable File System Interface For Linux," LinuxExpo 99, 1999.

[13] "Encrypting File System for Windows 2000," www.microsoft.com/WINDOWS2000/library/howitworks/security/.

[14] Evi Nemeth, Garth Snyder, Scott Seebass and Trent R. Hein, UNIX System AdministrationHandbook 3rd Edition, Prentice Hall, 2001.

[15] D. A. Thompson and J. S. Best, The future ofmagnetic data storage technology, IBM Journal ofResearch and Development, Vol.44, No.3, pp.311-322, May, 2000.

장 승 주

e-mail : sjjang@dongeui.ac.kr

1985년 부산대학교 계산통계학과(전산학) 학사

1991년 부산대학교 계산통계학과(전산학) 석사

1996년 부산대학교 컴퓨터공학과 박사

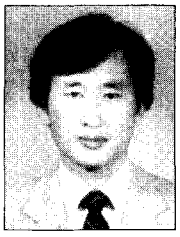
1987년~1996년 한국전자통신연구원 시스템 S/W연구실

1993년~1996년 부산대학교 시간강사

2000년~2002년 University of Missouri at Kansas City, visiting professor

1996년~현재 동의대학교 컴퓨터공학과 부교수

관심분야 : 운영체제, 분산시스템, Active Network, 시스템 보안



이 정 배

e-mail : jblee@sunmoon.ac.kr

1981년 경북대학교 전자공학과(공학사)

1983년 경북대학교 대학원 전자공학과 (공학석사)

1995년 한양대학교 대학원 전산공학과 (공학박사)

1982년~1991년 한국전자통신연구원 선임연구원

1996년~1997년 Dept. of Electrical & Computer Eng., U.C. Irvine 객원교수

1991년~2001년 부산외국어대학교 컴퓨터공학과 부교수

2002년~현재 선문대학교 컴퓨터정보학부 부교수

관심분야 : 컴퓨터네크워크, 실시간 시스템, 인터넷 응용