파일정보 중복을 이용한 제한된 사용자 수준 침입복구 기법*

류진형°, 박건형, 유병성, 이재국, 김형식
충남대학교 대학원 컴퓨터공학과
{ jinsto°, ghpark, feelluck }@cnu.ac.kr, {empire, hskim}@cs.cnu.ac.kr

A Limited User-level Intrusion Recovery Scheme Using Duplicated Information on Files

Jin-Hyong Ryu°, Gun-Hyong Park, Byung-Sung You, Hyong-shik Kim
Dept. of Computer Engineering, Chungnam National University

요 약

보안의 중요성이 높아짐에 따라 대응책 또한 관심이 높아지고 있다. 침입탐지와 침입방지 기법들이 나와 있지만 완벽한 보안체계 구축은 거의 불가능 하다고 알려져 있다. 침입복구 기법은 침입을 가정하고 침입당한 시스템의 파일들을 침입 이전의 상태로 돌려 시스템의 안정성 확보에 도움을 주는 방법이다. 본 논문에서는 관리자에 의해서 지정된 중요 파일의 변경내용을 주기적으로 저장(Archive)하여 복구(Recovery)할 수 있는 기법을 제안한다.

1. 서 론

인터넷 사용자의 증가와 더불어 악의 있는 사용자인 크래커도 많은 증가를 보이고 있다. 일반 사용자는 항상 신뢰성 있는 정보를 받아 보기 원하지만 크래커들은 서버에 침입하여 파일을 수정, 삭제함으로써 일반 사용자로 하여금 올바른 정보를 받아 보지 못하게 하고 있다. 그러므로 악의 있는 사용자의 침입에 의한 파일의 변경에도 올바른 정보를 일반 사용자에게 제공하기 위한 기술이 필요하게 되었다[1].

본 논문에서는 관리자에 의해 지정된 중요 파일들의 변경내용을 중복파일들로 저장하고 이들은 데몬 프로세스에 의해 주기적으로 관리된다. 지정된 중요 파일들이 침입에 의해 악의적으로 수정되거나 삭제되었을 때 중복파일들을 이용하여 파일의 정보를 복구하고 정보에 대한 무결성을 보장할 수 있는 방법을 제안하고 구현한다.

논문의 구성은 다음과 같다. 2절에서는 복구 시스템의 전체적인 구조를 설명하고 3절에서는 공간에 대한 추가적인 요구를 줄이고 복구의 효율성을 고려한 중복파일 생성 기법을 제안하고 4절에서는 생성된 중복파일을 관리하는 기법을 설명한다. 5절에서는 침입이 발생한 경우 생성된 중복파일을 이용하여 파일을 침입 이전 상태로 복구하는 기법을 제안한다. 6절에서는 본 논문에서 제안된 방법으로 실제 구현했을 때 구체적인 방법을 설명한다. 마지막으로 7절에서는 문제점과 향후 발전방향에 대해서 이야기한다.

2. 복구 시스템 구조

시스템은 크게 사용자 인터페이스, 응용처리, 파일시스템으로 나뉜다. 사용자 인터페이스계층은 시스템에 저장, 복구명령을 내리고 저장과 복구에 필요한 경로들을 생성하는 역할과 주기적으로 저장을 수행하는 데몬을 관리한다. 응용처리계층에서는 저장과 복구를 수행하고 파일시스템계층에서는 상위계층에서 원하는 중복파일의 데이터를 전달하고 저장하는 역할을 수행한다.

그림 1은 제안된 시스템의 구조를 도식화한 그림이다. 응용처리계층(Application Processing Layer)의 리눅스 파일시스템에는 복구 시스템에 필요한 옵션이나 관리할 파일들의 경로를 저장하는 리눅스 파일시스템이다. 파일시스템계층(File System Layer)에 정의된 파일시스템은 4절에서 제안한다. 이 파일시스템은 복구 시스템에서 제공하는 함수를 이용하여 상위 계층과 정보를 주고받는다. 화살표는 각각의 모듈들이 주고받는 방향과 데이터를 나타낸다.

* 본 연구는 "대학 IT 연구센터 육성지원사업"의 지원을 받아 수행한 연구 결과임.

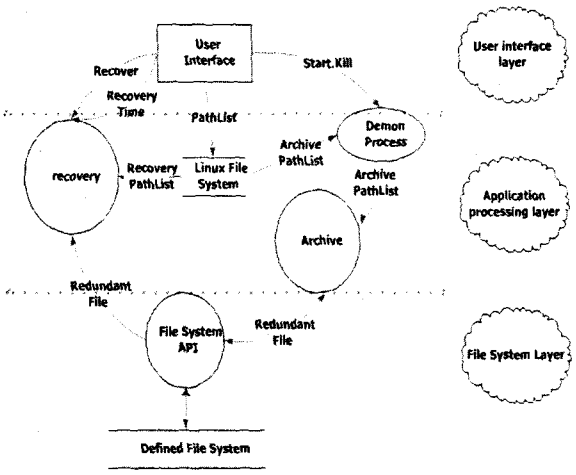

그림 1 복구 시스템 구조

3. 중복파일 생성

중복파일(duplicated file)이란 직전파일과 차분파일(differential file)을 통칭한다. 직전파일은 최근 저장시점의 원본파일을 말하고 차분파일은 직전파일과 현재 파일시스템의 원본파일과의 차분데이터를 저장하고 있는 파일을 지칭한다.

시스템에 임의의 파일 변경이 발생할 경우 무결성을 갖는 시점으로 복구되기 위해서는 원하는 시점에 대한 파일데이터 및 정보를 알아야 한다. 이 때 필요한 것들 중 하나가 차분파일이다. 차분파일은 저장 시점들 간의 파일의 변경된 데이터를 가지고 있어 복구할 때 시간 순서로 생성되어 있는 차분파일들을 직전파일에 역으로 적용시켜 파일을 복구한다.

차분파일의 형식은 GNU DiffUtil[2]의 형식 중 그림 2와 같은 형식을 사용한다.

```
@@ from-file-range to-file-range @@
line-from-either-file
line-from-either-file...
```

그림 2 차분파일 형식

- from-file-range: 이전 파일의 범위
- to-file-range: 현재 파일의 변경 범위
- line-from-either-file: 이전 파일과 비교하여 변경된 데이터를 줄 단위로 나타내며 각 줄의 첫 캐릭터는 + 또는 - 로 표시하고 + 는 추가된 내용 -는 제거된 내용을 표시

이러한 차분파일은 최근 저장시점 이후로 변경되지 않았다면 차분파일은 생성하지 않는다.

응용수준에서는 파일이 변경되는 순간을 알아내기 어렵기 때문에 파일 변경시간과 저장시간은 서로 다를 수 있다. 따라서 복구를 위해서는 차분파일과 함께 직전파일이 필요하다. 또한, 침입복구 시 복구 시점은 먼 과거보다 최근일 가능성이 높다. 파일을 과거시점에서 현재의 파일로 복구하기 보다는 현재의 파일에서 과거의 파일로 복구하는 것이 효과적이다. 이와 같은 이유로 직전파일의 시점은 최근 저장시점으로 정한다.

중복 파일들은 데몬에 의해 주기적으로 새성되며 최초 중복파일을 생성할 때에는 직전파일이 존재하지 않기 때문에 원본파일을 직전파일로 복사한다. 이후 중복파일을 생성할 때에는 직전파일과 원본파일을 비교하여 변경된 데이터만을 차분파일로 생성하고 원본파일을 다시 직전파일로 복사하지 않고 생성된 차분파일을 직전파일에 적용하여 직전파일을 갱신한다. 이렇게 하면 원본 파일을 복사하는 시간의 오버헤드를 감소시킬 수 있다. 이를 그림 3과 같이 도식화할 수 있다. 여기서 숫자는 발생하는 순서를 의미한다.

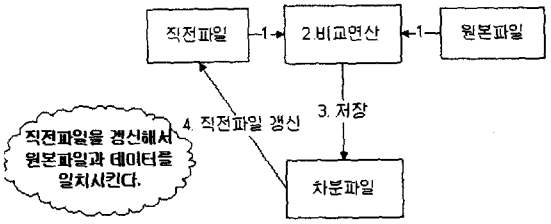

그림 3 중복파일 생성 순서

4. 중복파일 관리

중복파일은 데몬 프로세스에 의해 일정한 주기로 생성되므로, 하나의 원본파일에 여러 개의 중복파일이 생성된다. 따라서 직전파일과 별도로 차분파일들을 하나로 묶어 관리하는 기법을 제안한다. 그림 4는 차분파일들과 메타데이터들을 하나로 묶는 파일의 구조를 나타낸 것이다.

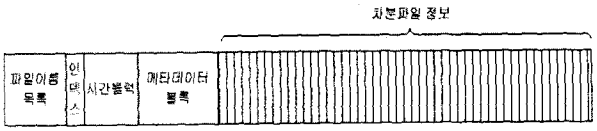

그림 4 차분파일 저장 구조

4.1 파일이름 목록 저장

그림 4와 같은 방법으로 차분파일을 저장하면 파일이름과 경로를 메타데이터 블록에 저장해야 한다. 파일 이

름과 경로를 저장하기 위하여 최신 리눅스 시스템에서 허용하는 최대한의 공간을 할당할 경우 4096바이트가 필요하다. 이 경우, 메타데이터의 공간 낭비가 발생하게 되어 다른 저장방법이 필요로 한다. 저장되는 파일 경로의 특징을 살펴보면 파일의 경로와 이름이 매번 중복된다는 것을 알 수 있다. 따라서 파일이름 목록을 생성하여 파일의 경로를 디렉토리 단위로 구분하여 고유한 숫자와 매치하여 저장하였다. 간단한 예를 들어 /var/log/message 파일을 저장한다면 표 1과 같은 형태의 파일이름 목록을 생성한다.

표 1 파일이름 목록

고유번호	이름
1	var
2	log
3	message
⋮	⋮

결과적으로 파일이름 목록과 1/2/3이라는 고유번호만 저장하게 된다. 이렇게 하면 매번 반복되는 이름들은 고유번호만 저장하면 되기 때문에 메타데이터의 공간을 줄일 수 있다.

4.2 메타데이터 관리

차분파일 저장 구조는 순환 형태의 구조를 가지며 공간이 부족할 경우 가장 과거의 정보를 지우게 된다. 이러한 순환구조는 시간 블록 처음과 시간 블록 끝을 인덱스 블록에 저장하는 방법으로 유지된다. 각각의 블록에서 순환 큐를 유지하는 방법을 도식화해서 살펴보면 그림 5와 같다.

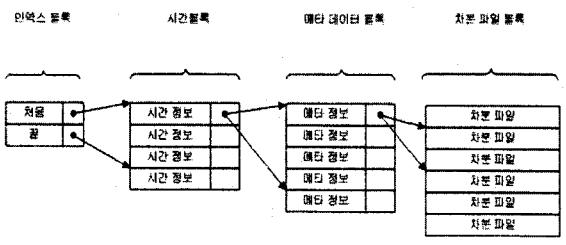

그림 5 순환 큐 유지 방법

각각의 블록들은 자신의 하위 블록의 포인터를 가진다. 이런 기법으로 처음과 끝을 유지하면 일관성을 가지는 순환 큐 구조를 유지할 수 있다. 또한 저장 공간이 부족할 경우 처음 인덱스를 증가시키면 가장 과거의 정보가 삭제되고 그 위에 최근 데이터가 저장될 수 있다.

일반적인 파일저장과 달리 차분정보의 저장은 시간이 유일한 키가 된다. 시간을 키로 메타정보와 차분파일을 구별하여 저장하면 데이터의 삽입, 삭제가 편리하다. 따라서 시간정보를 키로 메타정보 블록과 차분파일을 구분하여 저장한다.

제안된 메타데이터의 역할은 차분파일들을 찾아주는 역할과 원본파일에 대한 메타정보를 저장하여 복구 시 본래의 파일 속성을 유지시키는 것이다. 그림 6은 메타데이터에 저장되는 정보들의 구조를 나타낸 것이다.

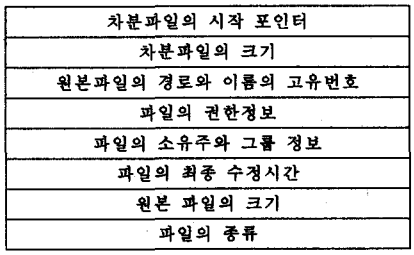

그림 6 메타데이터 구조

5. 침입 복구

손상된 파일을 복구하기 위하여 가장 최근에 저장된 직전파일과 차분파일 정보가 필요하다. 이러한 파일을 활용하여 원본파일을 복구하기 위해서는 차분파일들을 직전파일에 최근시점부터 원하는 시점까지 차례로 적용하는 것이다. 그림으로 나타내면 그림 7과 같다.

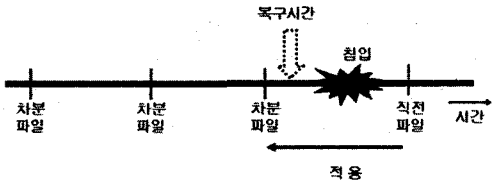

그림 7 침입복구 메커니즘

복구시스템은 복구하려는 파일이름에 대한 정보와 어느 시점으로 복구할 것인가를 나타내는 복구시점에 대한 정보를 활용한다. 이러한 정보를 얻기 위하여 침입 탐지 시스템과 같은 외부 시스템이나 관리자의 도움을 받는다. 그림 8은 복구기법을 위한 순서도이다.

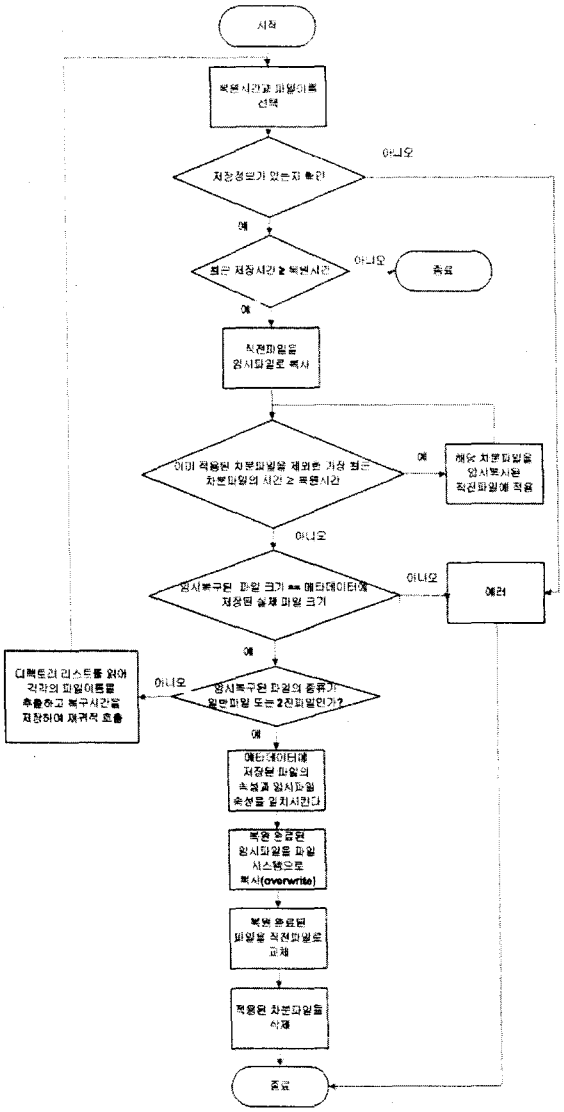

그림 8 복구 순서도

복구를 원하는 파일이름과 복구시점을 선택하고 이름정보로 저장정보가 존재하는지 확인하고 메타데이터를 읽어온다. 시간 블록에 저장된 차분파일 저장시간과 복원시간을 비교하여 복원시간이 시간상 앞서면 직전파일을 임시파일에 복사한다. 이 파일에 복원시간 보다 늦게 저장된 차분파일들을 시간의 역순으로 적용시킨다. 이렇게 하면 복원시점의 완전한 파일이 생성된다.

생성된 파일의 무결성을 입증하기 위하여 저장 시 파일 크기와 복원된 파일 크기를 비교한다. 만약 다르다면 파일을 복원하지 않고 에러를 기록한다.

복원될 파일이 디렉토라면 저장된 파일 이름정보를 하나씩 읽어와 복원파일 이름과 복원시점을 생성하여 재귀적으로 함수를 호출한다. 일반파일이나 이진파일이면 파일의 속성들을 메타 데이터에 저장된 속성과 일치시킨 후 원본 파일을 덮어 쓰는 방식으로 복사한다. 적용된 차분파일들은 침입 이후에 생성된 파일들이 되므로 삭제하고, 복원된 파일이 직전파일이 된다.

6. 구현

구현이 이루어진 시스템 환경은 표2와 같다.

표 2 구현 환경

CPU	펜티엄4 2.6GHz
Memory	1Gbyte
OS	Linux(Ubuntu)
Programming Language	C 언어, C++

사용자 인터페이스계층은 리눅스 QT 라이브러리[3]를 이용하여 구현했다. 응용처리계층과의 통신은 설정파일과 정의된 명령을 실행하는 방식을 사용했다. 라이브러리 특성상 C++언어를 이용하여 구현했다. 그림 9는 구현된 사용자 인터페이스 메인 다이얼로그를 보여준다.

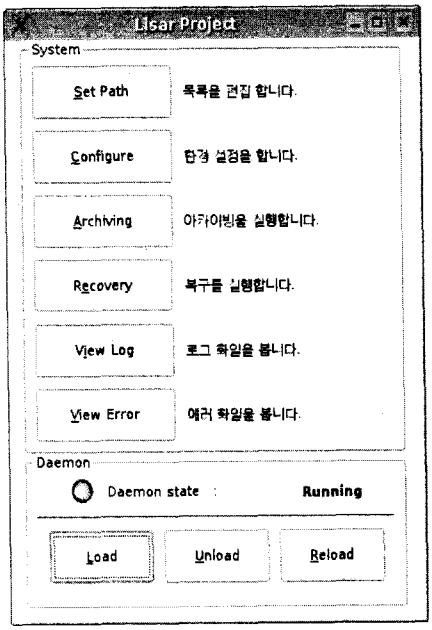

그림 9 구현된 사용자 인터페이스 메인 다이얼로그

함수 프로토 타입	설 명
int getArchiveTimes(time_t* archiveTime)	파일시스템에 저장되어 있는 차분파일들의 시점을 가져온다.
indexData readFileInfo(char* szPath, time_t archvieTime, infoData_t* pFileinfo);	파일이름과 시간정보를 이용하여 해당 파일의 정보를 가져온다.
int getFilePath(indexData indexOfFile, char* szPath);	파일의 경로와 이름의 고유번호를 이용하여 실제 파일시스템의 경로를 가져온다.
indexData getIndexOfPath(char* szPath);	파일의 경로를 이용하여 파일의 고유번호를 얻어온다.
int startOfArchive(time_t timeOfStart);	저장을 실행하기 전에 시간정보와 함께 함수를 호출하면 하나의 시간블록이 생성된다.
int endOfArchive();	모든 저장이 끝나면 함수를 호출한다.
int writePatchData(char* szPath , void* pData , int nSize, infoData_t* pFileinfo);	입력된 차분정보를 파일시스템에 쓴다.
int readPatchData(char* szPath, time_t archvieTime, void* pData, int nSize);	파일시스템에서 차분정보를 읽어온다.
int openPrevData(indexData indexOfFilePath);	직전파일을 여는 함수이다.
int closePrevData(int FileDescriptor);	직전파일을 닫는 함수이다.
int readPrevData(int FileDescriptor, void* pData, int nSize);	직전파일을 읽는 함수이다.
int writePrevData(int FileDescriptor, void* pData, int nSize);	직전파일을 쓰는 함수이다.
off_t readPrevFileSize(indexData indexOfFile);	직전파일의 크기를 알아오는 함수이다.

표 3 파일시스템 인터페이스 종류와 설명

응용처리 계층에서 데몬은 주기적으로 파일을 저장하는 역할만을 수행하고 실행주기와 저장파일의 목록을 지정된 파일에서 읽어오는 방식으로 구현했다.

복구와 저장에서는 LibXDiff[4]라이브러리를 프로그램에 맞게 수정 후 사용하여 차분파일을 생성한다. 디렉토리의 경우, 리눅스 파일시스템에서는 읽기가 금지되어 있기 때문에 디렉토리에서 저장된 파일들의 이름을 읽어와 새로운 파일에 저장하여 그 시점의 디렉토리에 어떤 파일들이 속해 있는지를 확인 가능하게 했다. 즉, 디렉토리는 포함하고 있는 파일의 이름을 저장한다. 이렇게 만들어진 디렉토리 파일도 이전 파일과의 차이점만을 저장하여 공간의 낭비를 줄인다.

현재의 파일시스템은 차분파일들은 하나의 파일로, 직전파일들은 여러 파일들로 존재한다. 하지만 상위계층에서는 정의된 응용 프로그래밍 인터페이스(API)로 차분파일과 직전파일을 이용한다. 이런 인터페이스의 종류와 설명은 표 3과 같다. 파일을 읽어오는 함수의 경우 파일의 크기를 읽어와 충분한 저장 공간을 확보한 후에 함수를 사용해야 한다.

7. 결론

본 논문은 악의적인 의도로 시스템에 침입하여 파일의 내용을 왜곡하였을 때 침입 이전의 파일로 복구하기 위해 연구되었다. 중요파일로 지정된 파일에 변경이 발생했을 때 중복파일을 저장하고 관리하는 기법을 제안하였으며 중복파일을 이용하여 시스템이 무결성을 갖는 시점으로 복구되기 위한 기법을 제안하고 응용수준에서 구현하였다.

현재의 파일시스템은 리눅스의 파일로 존재하여 시스템 관리자의 권한으로 볼 수 있는 문제점이 있지만 향후 파일시스템을 은닉할 수 있는 파일시스템으로 고안하여 개선할 계획이다.

참고문헌

[1] 이재국, 김형식, "리눅스 파일시스템에서의 로그기반 침입 복구 기법," 제30회 한국정보과학회 춘계발표회 논문집, 30권, 1호, 2004년 4월.

[2] GNU DiffUtil, http://www.gnu.org/software/diffutils/

[3] QTLib, http://www.trolltech.com/

[4] LibXDiff, http://www.xmailserver.org/xdiff-lib.html

[5] Marc J. Rochkind, "Advenced UNIX Programming", Addison Wesley, 2004년.