리눅스 파일시스템에서의 로그 기반 침입 복구 기법*

이재국° 김형식
충남대학교 컴퓨터과학과
{empire, hskim}@cs.cnu.ac.kr

A Log-Based Intrusion Recovery Scheme for the Linux File System

Jae-Kook Lee, Hyong-Shik Kim
Dept. of Computer Science, Chungnam National University

요 약

일반 사용자는 침입이 있더라도 항상 신뢰성 있는 정보를 획득하길 원하기 때문에 침입에 의하여 파일이 훼손되는 경우에도 사용자에게 투명한 방법으로 복구할 수 있는 방법이 필요하다.

본 논문에서는 리눅스 기반의 파일 시스템에서 변경이 일어날 때마다 로그 형태로 저장한 중복 파일을 이용하여 침입에 의하여 훼손된 부분을 복구하기 위한 기법을 제안한다. 실제 파일 변경이 일어날 때에는 그 변경이 악의적인지 합법적인지 판단이 불가능하다는 점에서 변경이 일어날 때마다 로그 형태로 변경정보를 유지하고 복구가 필요할 경우 선택적으로 적용할 수 있도록 하였다. 또한 로그 크기의 계속적인 증가에 효과적으로 대처하기 위한 로그 컴팩션(compaction) 방법도 보인다.

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 크래커들의 공격으로 파일이 손상을 입더라도 사용자에게 올바른 정보를 주기 위한 기술이 필요하게 되었다. 곧 침입으로 인하여 발생한 피해 파일시스템을 복구하는 기술이다.

우선 과거 버클리 대학에서 테입 형태의 저장 매체에 사용하였던 로그-구조 파일시스템(Log-structured File System)[1][2]에 대한 이해를 통하여 파일 복구 시스템의 복제될 데이터를 위한 새로운 로그 구조를 제안하며, 로그를 유지 관리하기 위한 컴팩션 방법도 제안한다.

2. 로그기반 침입복구

시스템이 침입에 의해 원본파일에 피해를 입었을 경우 복구하기 위하여 중복파일이 필요하다. 그런데 중복파일을 생성하는데 있어 원본파일에 변경이 일어날 때마다 새롭게 파일을 생성한다면 오버헤드가 너무 크다. 뿐만 아니라 원본파일이 크래커에 의해 손상을 입었다는 것을 나중에 알았을 때는 이에 중복파일도 동일하게 갱신되었기 때문에 실질적인 복구가 불가능하다. 만약 원본파일과는 별도로 변경이 일어난 부분만을 저장하는 로그 형태의 중복파일을 유지한다면 침입 복구를 위하여 효과적으로 활용할 수 있다. 그림 1에서와 같이 변경의 시점에서 원본파일은 변경이 일어나지만 중복을 위한 파일은 변경전의 내용을 수정하는 대신 변경내역만을 별도의 로그로 저장하기 때문에 침입이 발견되면 과거의 특정 시점으로 복구를 할 수 있다. 그 시점에 대한 결정은 시간 정보와 역할 정보를 활용하는데 사용자의 선택에 의존한다.

그림 1: 침입복구

3. 로그 구조

로그-구조 파일시스템(LFS)[1]은 추가 전용(append-only) 모드로 파일을 기록한다. 그러나 제안할 침입 복구 기법에서는 추가 전용 모드일 필요가 없다.

그림 2는 새롭게 정의한 침입 복구 시스템에서 로그에 대한 자료 구조를 나타낸다.

File Id
Time
Session ID
Current Offset
Next of End Offset
Size
Data

그림 2: 파일 복구 시스템을 위한 로그 구조

- File Id : 원본파일을 구별하기 위한 고유한 구별자
- Time : 로그가 기록된 시간, 복구 시간 정보로 활용
- Session Id : 현재 프로세스의 Session Id, 복구 역할 정보로 활용
- Current Offset : 파일 변경이 일어나는 시작 옵셋
- Next of End Offset : 파일 변경이 일어나는 끝의 다음 옵셋
- Size : 실제 수정이 일어난 크기
- Data : 변경된 내용

이 자료 구조는 파일에 대하여 일대일 수정이 일어나는 경우, 추가가 일어나는 경우, 삭제가 일어나는 경우에 모두 적용된다. 중복파일의 로그가 어떻게 기록되는지 그림 3의 예를 통하여 살펴본다. 파일을 중복하기 원하는 /dir/file1이

* 본 연구는 대학 IT 연구센터 육성/지원사업의 연구결과로 수행되었음

라는 파일이 세션 Id가 session1인 사용자에 의하여 첫번째로 수정이 일어나며 두번째로 세션 Id가 session2인 사용자에 의하여 데이터 추가가 일어난다고 하자. 그럼 파일 복구 시스템은 사용자에 투명하게 그림 3의 (b)와 같이 로그파일을 기록하게 된다. 그 내용을 분석해 보면 /dir/file1이라는 파일이 세션 Id가 session1인 사용자에 의하여 2월 20일 12시 53분에 현재 옵셋이 200인 곳에서 끝 다음 옵셋이 500인 곳까지 300바이트만큼 변경이 일어났으며 동일한 파일에 2월 20일 14시 50분에 세션 Id가 session2인 사용자가 파일의 추가가 일어나는 700인 곳에서부터 100바이트 만큼 데이터를 추가한 것을 알 수 있다.

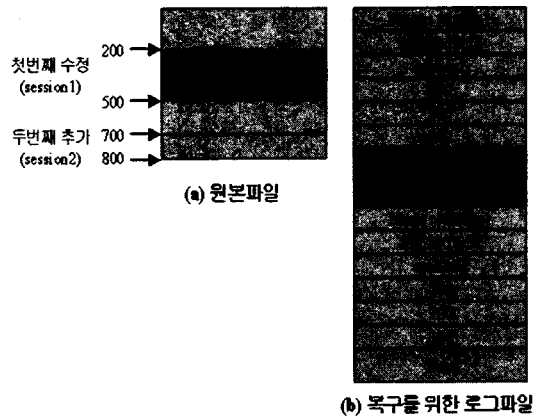

그림 3: 로그파일 저장 방법

삭제가 일어나는 경우에는 시작 옵셋과 끝나는 다음 옵셋을 기록하는 과정은 같다. 그러나 크기가 0이 된다.

4. 로그 컴팩션 메카니즘

침입 복구 시스템에서 컴팩션은 로그 형태로 저장되는 중복파일이 지나치게 커지는 것을 효과적으로 관리하기 위한 방법이다. 컴팩션을 적용할 경우 각 변경에서의 내역에 관한 정보가 상실되기 때문에 침입에 의하여 훼손되지 않았다는 것을 보장할 수 있을 경우에만 컴팩션 기능을 적용할 수 있다.

예를 들어, 그림 4는 로그파일로부터 컴팩션 하는 방법이다. 파일(/dir/file1)에 대한 변경으로 로그가 다음과 같이 기록되었다고 하자. (a)의 로그1은 200 옵셋 위치에서 200바이트만큼 추가를 한 것이고, (b)의 로그2는 500 옵셋 위치에서 200바이트만큼 수정하였다. 그리고 (c)의 로그3은 300 옵셋 위치에서 300바이트만큼 수정한 것이다. 변경 내용을 로그가 갖고 있다가 컴팩션이 일어나게 되면 200 옵셋 위치에서 299 옵셋 위치까지는 로그1의 데이터1이 들어가게 되고 300 옵셋 위치에서 599 옵셋 위치까지는 로그3의 데이터3이 들어가고, 600 옵셋 위치에서 끝까지는 로그2의 데이터2가 들어가게 된다. 결과적으로 200 옵셋 위치에서 500바이트 변경이 일어난 것으로 컴팩션되어 로그파일에 나타난다. (d)의 로그헤더부분을 보면 'Next of End Offset'의 값이 500인 것을 볼 수 있다. 이것은 (a)의 로그1에서 추가가 일어났기 때문에 로그1 이전의 시점에서 보면 200 옵셋 위치부터 300바이트의 데이터가 저장되었던 공간에 500바이트가 추가된 것으로 간주되기 때문이다.

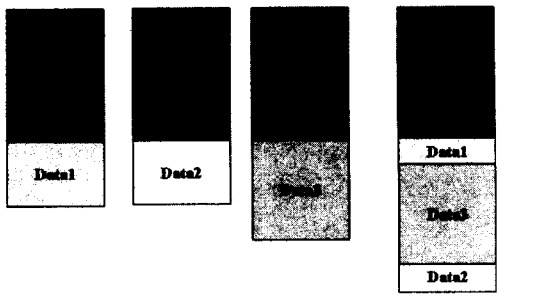

그림 4: 로그 컴팩션

5. 설계

5.1 시스템 호출 재작성

응용수준에서 파일을 작성하기 위하여 사용되는 open(), write(), close()등의 시스템 호출은 커널수준에서 sys_open(), sys_write(), sys_close() 과 같은 시스템 호출을 하게 된다 [3]. 본 논문에서는 침입 복구 기법을 구현하기 위하여 시스템 호출 테이블의 관련 함수들을 변경하고 응용프로그램에서 호출이 발생할 때, 새롭게 작성된 시스템 호출 함수들이 호출되도록 하였다.

(1) new_sys_open()

응용수준에서 파일의 기술자를 얻기 위하여 open() 라이브러리 함수 요청이 있을 경우 커널수준에서는 sys_open() 시스템 호출을 하게 된다. 이 과정에서 중간에 시스템 호출을 가로채어 로그파일에 대한 파일 기술자와 관련하여서 사용자에 투명하게 sys_open()을 해주도록 모듈 프로그램으로 재구성해야한다. 즉 재작성된 new_sys_open()은 원본파일에 관하여 sys_open() 시스템 호출을 하고, 사용자에 투명하게 로그파일과 관련하여서도 sys_open()한다. new_sys_open()의 전반적인 수행 과정은 그림 5와 같다.

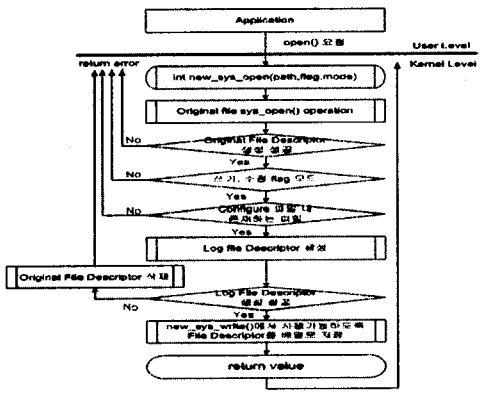

그림 5: new_sys_open() 처리과정

(2) new_sys_write()

파일 복구 시스템을 위하여 재작성된 new_sys_write() 시스템 호출은 new_sys_open() 시스템 호출에 의하여 생성된 로그파일 관련 파일 기술자에 파일시스템 복구를 위한 로그의 헤더 자료구조 값을 추가한 후 변경된 버퍼의 내용과 함께 sys_write()해야 한다. 그러나 원본파일 sys_write() 과정에서 에러가 발생한다면 원본파일의 sys_write() 반환 값으

로 로그파일 헤더의 크기 부분을 바꾸어준다. new_sys_write()의 전반적인 수행 과정은 그림 6과 같다.

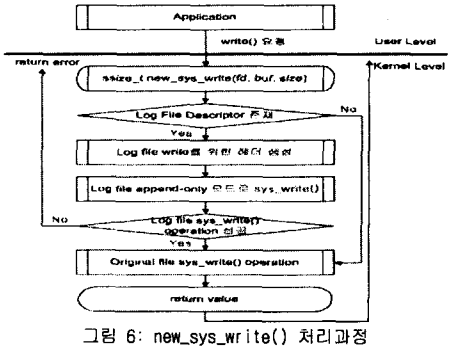

그림 6: new_sys_write() 처리과정

(3) new_sys_close()

재작성된 new_sys_close() 시스템 호출은 응용수준에서 close()를 호출하였을 경우 커널수준에서의 일반적인 sys_close() 시스템 호출과 비슷하다. 여기서는 원본파일에 대한 파일 기술자뿐만 아니라 중복 파일에 대한 파일 기술자도 반환하도록 구성된다. new_sys_close()의 전반적인 수행 과정은 그림 7과 같다.

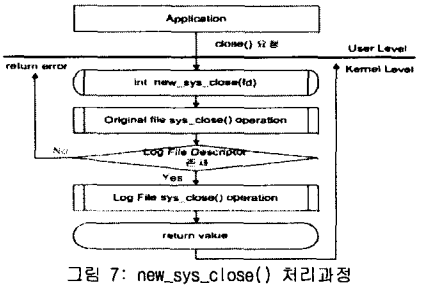

그림 7: new_sys_close() 처리과정

5.2 파일 복구 메카니즘

사용자에 투명하게 그림 8과 같이 로그가 기록되어 있다고 하자. L4 로그 기록 후 침입이 발생하여 파일을 복구하기 원한다고 하자. 이때 복구 스팬(recovery span)을 침입이 일어난 시점으로부터 복구 기준 시점까지의 범위로 정의한다. 복구 윈도우(recovery window)는 통상 침입 시점에서 복구 스팬 범위 전부이지만 복구 스팬 범위 내에 컴팩션이 있었을 경우에는 침입시점부터 컴팩션 시점까지로 정의 되며 내부의 로그에 대하여 신뢰 여부를 알 수 없음을 표시한다. 즉 복구 윈도우의 크기는 복구스팬과 직전의 컴팩션 시점에 의하여 결정된다. 그림 8에서는 로그 L4가 복구 윈도우 내에 있기 때문에 L4의 신뢰 여부를 알 수 없다.

결국 로그의 시간 정보를 활용하여 복구 윈도우를 벗어나는 것은 복구 대상으로 하고 복구 윈도우 내부에 있는 것은 로그의 역할 정보를 이용하여 복구를 한다.

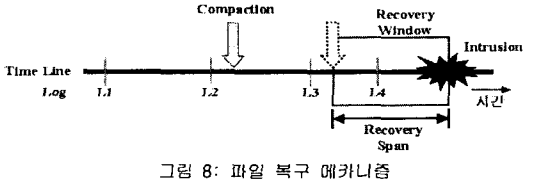

그림 8: 파일 복구 메카니즘

5.3 시간 정보의 활용

시간 정보를 활용하여 파일 복구가 일어나는 과정을 그림 8을 통하여 보자. 컴팩션은 12시 53분 이전에 일어났다고 하고, 복구 시스템에 오후 3시에 침입이 있었다고 하자. 복구 시스템은 로그를 확인한다. 동일한 파일 구별자를 갖는 두개의 로그는 시간 정보에 의하여 선택적으로 복구에 이용된다. 복구스팬 범위가 30분이라고 할 경우, 이 경우에는 로그 윈도우도 2시30분부터 3시까지로 복구스팬과 크기가 같다. 결국 첫번째 로그는 무조건적인 복구의 대상이 된다. 반면 두 번째 로그는 로그 윈도우 내부에 위치하므로 역할 정보를 활용하여 복구 대상이 되는지를 결정한다.

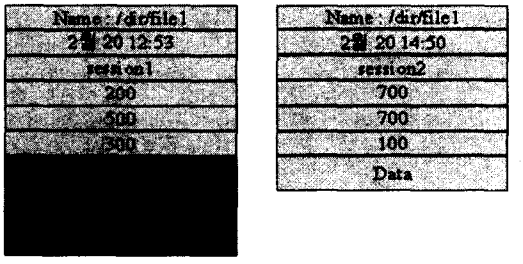

그림 9: 시간/역할 정보 활용에 대한 예

5.4 역할 정보의 활용

역할 정보를 활용하기 위하여 로그 자료구조는 세션 Id를 사용하게 된다. 그림 9에서 원본파일 /dir/file1의 파일의 소유자의 세션 Id가 sesion1이라고 할 때, 전반적인 침입 복구 과정은 시간 정보를 활용한 방법과 같다. 그러나 복구 윈도우 내부에 있는 로그의 경우 로그에 기록된 세션 Id와 로그인 사용자의 세션 Id를 비교하여 신뢰성 있는 사용자 인지를 확인하여 로그를 복구의 대상으로 삼는다. 5.3절의 예에서처럼 로그 윈도우 범위 안에 있는 두 번째 로그의 세션 Id를 확인한다. 만약 두 번째 로그의 세션 Id session2를 신뢰할 수 없다면 두 번째 로그는 복구의 대상이 되지 않는다.

6. 결 론

본 논문에서는 침입 복구를 위한 파일 시스템 구성에 필요한 로그 구조를 설계하였고, 로그의 유지 관리를 위해 컴패션 방법을 제안하였다. 또한 침입 복구를 위해 복구 윈도우 밖의 로그는 시간 정보를 활용하여 복구하며 내부의 로그는 역할 정보를 활용하여 복구하는 방법을 제안하였고, 커널수준에서 기존의 시스템 호출을 대체하는 방법으로 설계하였다.

현재 각 기능에 대한 커널수준 검증이 이뤄졌으며 향후 전체 기능에 대한 통합 적용을 통하여 문제점을 개선할 예정이며, 효과적인 로그 컴팩션 방법을 통해 성능에 미치는 영향을 최소화할 계획이다.

참고문헌

[1] Mendel Rosenblum, "The Design and Implementation of A Log-structured File System", Kluwer Academic Publishers, 1995.

[2] Mendel Rosenblum, John K. Ousterhout, "The Design and Implementation of a Log-structured File System", Electrical Engineering and Computer Sciences, Computer Science Division University of California Berkeley, 1991.

[3] W. Richard Stevens, "Advanced Programming in the UNIX Environment", Addison Wesley, 2001.