로그기반 침입복구모듈을 위한 링크 정보 관리 기법*

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

A Link Information Management Scheme for the Log-Based Intrusion Recovery Module

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

요 약

악의 있는 해커들은 악성 프로그램을 이용하여 시스템에 침입하고 파일을 변경(추가, 수정, 삭제)함으로써 일반 사용자로 하여금 올바른 정보를 받아 보지 못하게 한다. 그러나 침입이 있더라도 사용자에게 신뢰성 있는 정보를 제공하기 위하여 로그기반 침입복구모듈을 제안하고 구현하였다. 구현된 로그기반 침입복구모듈은 복구를 위해 사용될 로그를 관리하기 위하여 많은 오버헤드가 발생했다..

본 논문에서는 로그기반 침입복구모듈의 성능을 개선하기 위하여 링크 정보를 관리하는 방법을 제안한다. 그리고 개선된 로그기반 침입복구모듈의 성능을 측정하여 변경 전과 비교한다.

1. 서 론

악의 있는 해커들은 악성 프로그램을 이용하여 시스템에 침입하고 파일을 변경(추가, 수정, 삭제)함으로써 일반 사용자로 하여금 올바른 정보를 받아 보지 못하게 한다. 이러한 공격으로부터 파일시스템이 훼손되는 것을 막기 위해 침입탐지시스템(IDS), 침입차단시스템(Firewall)과 같은 보안 솔루션들이 제안되었다. 그러나 침입탐지시스템은 기존에 발생한 침입탐지패턴의 소유 여부에 따라 침입탐지 여부가 결정되기 때문에 새로운 침입패턴의 공격일 경우는 탐지할 수 없다. 그리고 침입차단시스템은 허가된 패킷만 통과하므로 허가된 서비스의 취약성을 이용한 공격이나 내부자에 의한 공격에 취약하다. 이들 보안 솔루션들의 한계점으로 인하여 파일시스템이 훼손될 경우 피해 파일시스템을 침입 이전으로 복구하는 침입복구기술이 필요하다.

로그기반 침입복구모듈은 리눅스 파일시스템에서 특정파일에 변경이 일어나면 변경된 데이터에 대한 로그를 사용자에게 투명한 방법으로 기록한 후 침입에 의해 파일이 훼손될 경우 기록한 로그를 이용하여 침입이전으로 복구한다[1][2].

* 본 연구는 대학 IT 연구센터 육성/지원사업의 연구결과로 수행되었음

그러나 개발된 로그기반 침입복구모듈은 링크 정보를 관리하기 위하여 많은 오버헤드가 발생한다. 링크 정보를 관리하기 가장 쉬운 방법은 중복된 파일처럼 각각의 파일에 대하여 로그를 관리하는 것이다. 그러나 여러 개의 파일이 링크되어 있는 경우 링크를 해제할 때 마다 새롭게 로그를 기록하는 것은 저장 공간을 낭비하게 되고, 로그 기록을 위해 시간을 소비하게 된다. 뿐만 아니라 리눅스 파일시스템은 여러 개의 링크된 파일이 하나의 i-node를 갖는데, 링크된 파일마다 로그파일을 갖게 하는 것은 일관성이 없다.

본 논문에서는 성능 개선을 위하여 링크 정보를 관리하기 위한 새로운 방법을 제안하고, 개선된 로그기반 침입복구 모듈의 성능을 측정한다.

2. 설정파일관리

로그기반 침입복구모듈은 리눅스 파일시스템에서 특정파일에 변경이 일어나면 변경된 데이터에 대한 로그를 사용자에게 투명한 방법으로 기록한 후 침입에 의해 파일이 훼손될 경우 기록한 로그를 이용하여 침입이전으로 복구한다[1][2]. 로그기반 침입복구모듈은 전체 파일을 대상으로 로그를 기록하는 것이 아니라 특정파일에 대하여 선택적으로 로그파일을 생성했다. 그리고 링크 정보를 관리하기 위하여 파일의 이름으로 로그 정보를 관리했다. 이와 같은 이유로 이전 모듈에서는 표 1과 같은 이중 연결 리스트로 설정파일을 구현하였다.

표 1 설정파일을 위한 이중 연결 리스트 구조

```
struct conf_list {
        struct list_head cl_link;
        unsinged long conf_ino;
        const char *conf_fname;
}
```

그러나 이 자료구조는 open()에서 로그의 생성여부를 확인하기 위하여 설정파일을 검색하는데 많은 오버헤드를 발생시킨다. 그래서 오버헤드를 줄이면서 설정파일을 관리하기 위하여 리눅스 파일 시스템의 i-node 번호에 기반 한 배열 구조로 개선한다. 그리고 로그 구조에 파일의 이름을 구별하기 위하여 '파일식별자'를 추가한다. 변경된 로그의 구조는 그림 1과 같다. 기존의 자료구조에 추가적으로 'File Path Id'라는 파일식별자가 추가된다.

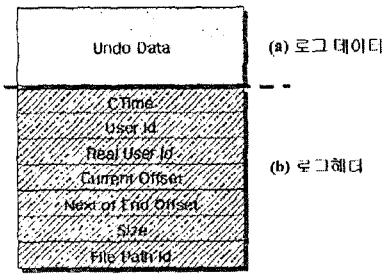

그림 1 로그구조

3. 트래쉬 기법

리눅스 파일시스템에서 특정파일을 변경하기 위해서 open(), write(), close()의 파일연산을 수행한다. open()에서 파일기술자를 획득하고 획득한 파일기술자를 매개변수로 하여 write() 연산을 수행하여 데이터를 변경하고 close() 연산을 통해 파일기술자를 반환하여 재사용하도록 하는 일련의 과정을 거친다. 이 과정에서 사용자에게 투명하게 로그를 기록하기 위해서 변경하고자 하는 원본파일을 open()하는 시점에서 로그파일도 같이 open()한다. 원본파일이 write() 연산에 의해 변경되기 전에 침입복구모듈은 변경되는 부분의 이전 데이터를 로그파일에 기록한다. 이렇게 사용자에 투명하게 open()과 write() 연산이 끝나면 close() 연산을 통하여 원본파일과 로그파일의 파일기술자를 반환하여 재사용가능하게 한다. 이렇게 로그를 관리하기 위한 연산을 리눅스 시스템의 파일연산 시스템 호출을 가로채어 랩핑(Wrapping)한다. 이 외에 리눅스의 unlink() 시스템 호출은 파일의 링크 카운트를 감소시키고 파일을 제거한다.

만약 여러 개의 파일이 링크되어 있다면, 침입복구모듈은 링크를 해제할 때마다 로그를 간직해야 한다. 그러나 하드링크(hard link)되어 있는 파일은 i-node 번호가 일치하므로, i-node 번호를 로그파일의 이름으로 갖는 로그기반 침입복구 모듈에서는 같은 로그파일에 로그가 기록되게 되어 일관성(consistency)을 유지할 수 없다. 또한 동일한 데이터를 여러 번 기록하여 저장공간을 낭비할 수 있다. 이러한 문제를 해결하기 위하여 트래쉬(trash) 기법을 제안한다. 링크가 여러 개 연결되어 있는 파일이 unlink() 시스템 호출을 이용하여 파일의 링크 카운트를 감소할 경우 바로 링크를 해제 아니라 트래쉬 디렉토리에 링크를 새롭게 연결해 주고, 신뢰성을 갖는 구간에 대하여 퍼지[2]를 할 때 실질적으로 트래쉬 디렉토리에 있는 링크를 해제하여 원본파일의 링크 카운트를 감소시킨다. 예를 들어 <그림>과 같이 3개의 파일이 링크되어 있다고 하자.

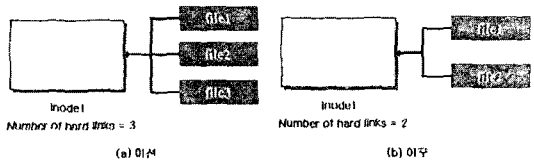

그림 2 unlink() 호출

unlink() 시스템 호출을 하게 되면 그림 2의 (b)와 같이 링크 카운트가 감소하고 파일이 제거되나, 트래쉬 기법을 이용하여 그림 3과 같이 트래쉬 디렉토리에 새로운 링크를 연결한다.

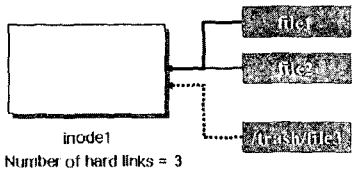

그림 3 트래쉬기법

그러나 링크가 해제된 파일은 표 2와 같은 자료 구조를 통하여 링크가 해제된 것을 시스템이 알 수 있도록 한다. 이를 이용하여 로그를 유지하고, 복구할 때 트래쉬 디렉토리의 링크를 이용할 수 있게 한다.

표 2 링크 해제된 파일 리스트 구조

```
struct trash_list {
        struct list_head tl_link;
        unsinged long trash_ino;
        const char *conf_fname;
        time_t trash_time;
}
```

4. unlink() 시스템 호출 재작성

unlink() 시스템 호출은 매개변수로 파일이름을 받아 링크 카운트를 감소시킨다. 만약 링크카운트가 0이 되고, 어떤 프로세스도 파일을 열어두지 않았다면 파일은 삭제된다. unlink 시스템 호출을 가로채어 트래쉬 기법을 적용한 sys_unlink() 시스템 호출의 순서도는 그림 4와 같다. 매개변수 pathname을 이용하여 i-node 번호를 획득하고 로그 기록 여부를 확인하고 로그를 기록해야 할 경우 트래쉬 디렉토리로 링크를 연결한다. 이 과정에서 트래쉬 리스트를 작성한다. 원본파일의 unlink()가 에러 없이 성공하게 되면 0을 반환하게 된다.

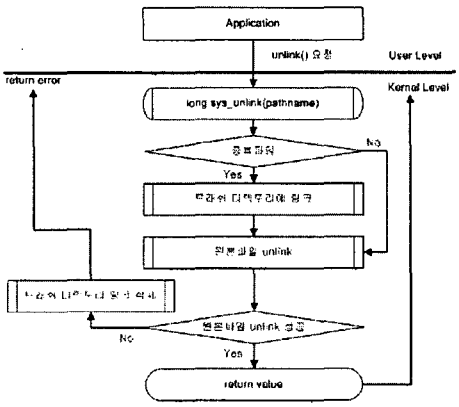

그림 4 unlink() 시스템 호출 순서도

5. 성능분석

그림 5는 침입복구모듈을 새롭게 구현하기 전과 후의 1k 바이트의 크기를 갖는 파일을 수정하는데 소요되는 시간(μs)을 측정한 결과이다. open 시스템 호출의 경우 변경 전(그림 5의 'before')에 비해 오버헤드가 약 22% 정도 감소하고, write 시스템 호출의 경우 약 5% 정도 증가한다.

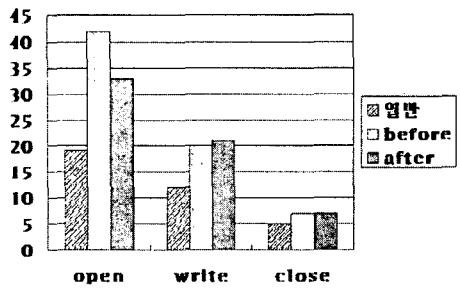

그림 5 성능분석

open() 시스템 호출에서 오버헤드가 감소한 부분, 즉 로그가 생성되고 관리되어야 하는지의 여부를 판단하기 위한 검색 시간만을 측정하면 그림 6과 같이 변경 전(그림 6의 'before') 이중 연결 리스트를 이용할 때 보다 약 75% 감소한다.

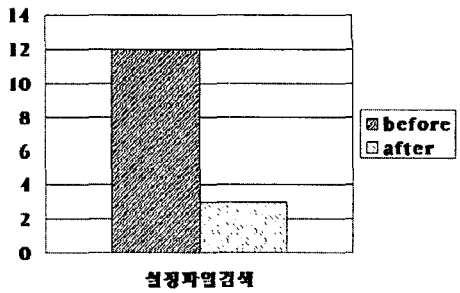

그림 6 설정파일검색 성능분석

6. 결론 및 향후 연구 과제

본 논문에서는 로그기반 침입복구모듈의 성능 개선을 위하여 설정파일관리 구조와 트래쉬 기법을 이용한 링크 정보 관리 방법을 제안하였다. 제안된 모듈을 위해 필요한 자료구조를 변경하였으며 unlink() 시스템 호출을 추가적으로 재구성하였다. 끝으로 모듈을 구현하여 실제 시스템에 적용함으로써 변경 전 침입복구모듈과의 성능을 비교하였다. 성능을 측정한 결과 로그를 관리하기 위하여 발생하는 오버헤드가 감소한 것을 확인하였다.

앞으로 생성된 로그파일과 설정파일을 보다 안전하게 저장하기 위한 저장 메커니즘에 대한 연구를 계속할 계획이다.

참고문헌

[1] 이재국, 김형식, "리눅스 파일시스템에서의 로그 기반 침입 복구 기법," 한국정보과학회 2003년 봄 학술발표논문집(A), pp. 413-415, 2003년 4월.

[2] 이재국, 김형식, "리눅스 파일시스템을 위한 로그 기반 침입 복구 모듈의 구현," 한국정보과학회 2004년 봄 학술발표논문집(A), pp.244-246, 2004년 4월.

[3] Rechard Stones and Neil Matthew, Beginning Linux Programming, 2nd Ed. p135~182, 정보문화사, 2000년 4월.

[4] Simon S.Y. Sbim, Li Gong, Aviel D. Rubin, Linley Gwennap, "Securing the High-Speed Internet," IEEE, Computer, Volume 37, Number 6, pp.33-35, June 2004.

[5] 최중섭, "Intrusion Tolerant System:An Introduction and a Simple Example," NETSEC-KR 2002, 제8회 정보통신망 정보보호 워크숍, pp.885~897, 16~17 May 2002.

[6] Alfonso Valdes, Magnus Almgren, Steven Cheung, Yves Deswarte, "An Adaptive Intrusion-Tolerant Server," LNCS, Volume 2845, pp.158~178, 10 Dec. 2003.