센서 기반 침입 탐지 시스템의 설계와 구현

최 종 무[†] · 조 성 제[††]

요 약

컴퓨터 시스템에 저장된 정보는 불법적인 접근, 악의적인 파괴 및 변경, 우연적인 불일치 등으로부터 보호되어야 한다. 본 논문에서는 이러한 공격을 탐지하고 방어할 수 있는 센서기반 침입탐지시스템을 제안한다. 제안된 시스템은 각 중요 디렉터리에 센서 파일을, 각 중요 파일에 센서 데이터를 설치한다. 이들 센서 객체는 일종의 덫으로서, 센서 객체에 대한 접근은 침입이라고 간주된다. 이를 통해 불법적으로 정보를 복사하거나 빼내 가려는 가로채기 위협을 효과적으로 방어할 수 있다. 제안된 시스템은 리눅스 시스템 상에서 적재 가능한 커널 모듈(LKM: Loadable Kernel Module) 방식을 사용하여 구현되었다. 본 시스템은 폭 넓은 침입탐지를 위해 호스트 기반의 탐지 기법과 네트워크 기반의 탐지 기법을 서로 결합함으로써 잘 알려지지 않은 가로채기 공격도 탐지가능하게 하였다.

키워드 : 침입 탐지 시스템, 센서 파일, 센서 데이터, 리눅스, 모듈

Design and Implementation of Sensor based Intrusion Detection System

Jongmoo Choi[†] · Seongje Cho[††]

ABSTRACT

The information stored in the computer system needs to be protected from unauthorized access, malicious destruction or alteration, and accidental inconsistency. In this paper, we propose an intrusion detection system based on sensor concept for detecting and preventing malicious attacks. We use software sensor objects which consist of sensor file for each important directory and sensor data for each secret file. Every sensor object is a sort of trap against the attack and it's touch can be considered as an intrusion. The proposed system is a new challenge of setting up traps against most interception threats that try to copy or read illicitly programs or data. We have implemented the proposed system on the Linux operating system using loadable kernel module technique. The proposed system combines host-based detection approach and network-based one to achieve reasonably complete coverage, which makes it possible to detect unknown interception threats.

Key Words : Intrusion Detection System, Sensor File, Sensor Data, Linux, Module

1. 서 론

인터넷(Internet)은 수많은 네트워크들을 연결하여 전 세계의 사람들이 서로 통신할 수 있는 서비스를 제공하고 있다. 인터넷은 전 세계에 있는 네트워크와 컴퓨터들을 하나의 표준으로 연결하는 개방형 네트워크로 이러한 개방성은 네트워크의 네트워크로서 성공할 수 있게 하여준 중요한 특성이지만, 다양한 정보보호 취약점을 가지고 있다[7]. 이러한 취약점을 이용하여 공격자는 자신이 목표로 한 시스템 및 네트워크에 대한 정보를 수집하고, 수집된 정보를 이용하여 시스템에 침입하게 된다.

침입(Intrusion)이란 "자원의 무결성(Integrity), 기밀성(Confidentiality), 가용성(Availability), 인증(Authentication)을 훼손시키려는 일련의 행동"으로 정의 된다. 공격자는 트로이 코드(Trojan code), 컴퓨터 바이러스(Virus), 웜(Worm)과 같은 악성 프로그램, 서버의 기능의 저하시키거나 무력하게 만드는 서비스 거부(DoS) 공격, LAN을 통과하는 패킷을 들여다보는 스니핑(Sniffing), 다른 주소로 위장하는 ARP 스푸핑(Spoofing), IP 스푸핑, DNS 스푸핑 등 많은 위협과 공격 방법으로 시스템에 침입을 시도하게 된다[8].

공격자의 이러한 위협 및 침입 시도로부터 중요한 자료와 시스템을 보호하기 위해 사용자는 침입차단시스템(방화벽)과 침입탐지시스템을 이용한다. 침입차단시스템은 불법적인 접근이나 공격으로부터 내부 네트워크를 방어하기 위해 내부 네트워크와 외부 네트워크 사이의 통로에 설치된다. 침입탐지시스템은 내부 네트워크 및 호스트에 설치되어 침입을 탐지하게 된다. 침입차단시스템은 주로 패킷 필터링 기술을 이용하며, 이 경우 패킷의 내용까지는 검사하지 않으므로 바이러스

※ 이 연구는 2003학년도 단국대학교 대학연구비의 지원으로 연구되었음.
† 종신회원 : 단국대학교 정보컴퓨터학부 조교수
†† 종신회원 : 단국대학교 정보컴퓨터학부 부교수
논문접수 : 2005년 5월 27일, 심사완료 : 2005년 9월 1일

에 감염된 프로그램이 압축되어 전송되면 침입차단시스템을 통과할 수 있다. 반면 침입탐지시스템은 패턴비교기술을 이용하며, 따라서 침입 패턴이 존재하는 경우 즉 알려진 침입에 대하여 탐지는 가능하지만 새로운 침입의 경우 탐지가 어렵다는 단점이 있다[8].

본 논문에서는 새로운 침입 시도에 대해서도 탐지가 가능한 새로운 형태의 침입탐지시스템인 SIDS(Sensor based Intrusion Detection System)를 제안한다. SIDS는 침입을 탐지하기 위한 소프트웨어 객체로 "센서"를 사용한다. 센서는 디렉터리를 보호하는 "센서 파일"과 중요 파일을 보호하는 "센서 데이터" 2가지가 있다. 시스템 관리자만이 접근, 관리할 수 있는 소프트웨어 객체인 센서를 사용함으로서, 센서를 접근하는 모든 침입에 대한 탐지가 가능하다. 또한 침입을 탐지한 후에는 중요한 자료의 유출 및 삭제와 같은 작업을 시스템 수준에서 방지하여 피해를 최소화할 수 있다.

SIDS는 호스트 기반의 침입 탐지기법과 네트워크 기반의 침입 탐지 기법을 결합하여, 체계적으로 침입을 탐지 및 대처할 수 있다. 제안된 시스템은 리눅스 상에서 LKM(Loadable Kernel Module)방식을 사용하여 구현되어 손쉬운 설치 및 성능향상을 제공하며, 새로운 침입을 발견하고 프로그램 및 데이터의 삭제, 변경과 같은 피해를 최소화 할 수 있도록 고려되었다.

본 논문의 구성은 다음과 같다. 2장에서는 침입탐지시스템에 대한 관련 연구 내용을 조사한다. 3장에서는 "센서 파일"과 "센서 데이터"를 사용한 SIDS의 모델을 제안하고 4장에서 SIDS에 대한 설계 및 구현에 대해 설명한다. 5장에서는 구현된 SIDS를 실험하고 성능을 측정했으며, 6장에서 결론 및 향후 연구방향에 대해 기술한다.

2. 관련 연구

본 장에서는 침입탐지에 사용되는 기술 및 침입탐지시스템(IDS : Intrusion Detection System)의 종류에 대하여 알아본다. 침입탐지시스템은 정보시스템의 비밀성, 무결성, 가용성, 인증에 피해를 주는 모든 상황을 탐지하는 것을 목표로 개발된 시스템이다. 따라서 침입탐지시스템은 침입자에 의한 불법적인 사용뿐만 아니라 합법적인 사용자에 의한 오용이나 남용도 찾을 수 있어야 한다. 또한 침입의 가능성이 있을 때에는 침입을 가능한 한 빨리 탐지하는 것뿐만 아니라 이에 적절히 대응할 수 있어야 한다.

침입탐지 모델에 따라 탐지 방법은 크게 오용침입(misuse intrusion) 탐지와 이상침입(anomalous intrusion) 탐지로 구분된다[2, 8]. 일반적으로 침입이라고 정해진 모델과 일치하는 경우를 탐지하는 것을 오용침입 탐지라고 하며, 접근 시 정해진 모델을 벗어나는 비정상행위를 탐지하는 것을 이상침입 탐지라고 한다.

오용침입 탐지는 정보시스템의 한 취약점에서 실제로 침입의 문제가 발생하는 경우 이를 탐지하는 기능으로 침입 사고가 일어날 때 나타나는 정형적인 패턴들을 감시함으로써 탐지가 이루어진다[14, 15]. 이를 위해 침입탐지시스템은 다양한 침입 사고의 패턴들을 사전에 수집하고 이러한 패턴의 상황이 발생하는지를 감시한다. 이미 알려진 침입형태를 패턴으로 코드화 한 후 이 코드를 발생하는 사건들의 패턴과 비교하여 침입여부를 판단하는 패턴 매칭 방식은 이미 알려진 취약성에 대해서는 침입탐지가 가능하지만 발견되지 않은 형태의 침입은 발견하기 어렵다.

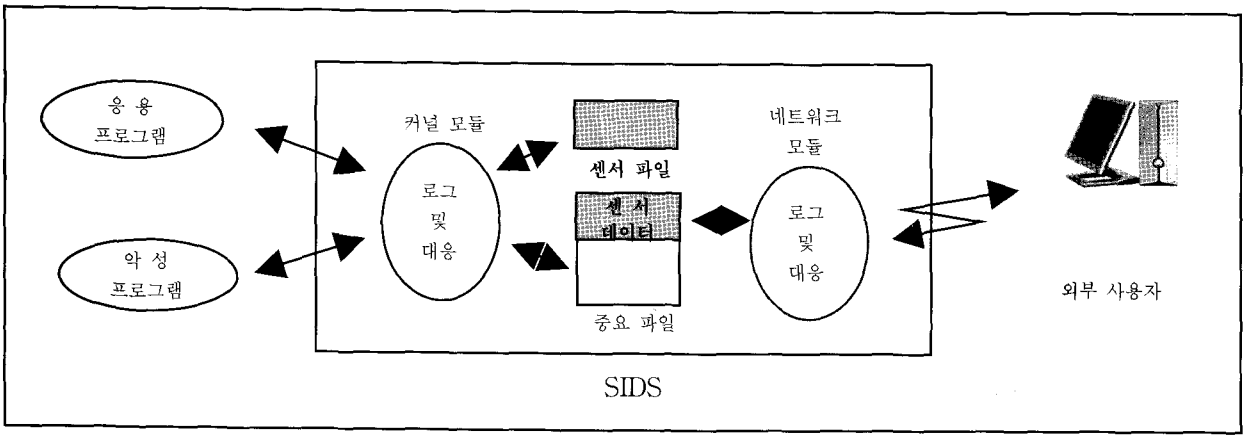
이상침입 탐지는 실제적인 외부의 침입이라기보다는 일반적인 시스템 사용 패턴에서 벗어나는 행위들을 탐지하는 것을 말한다[16, 17]. 이러한 비정상적인 행위는 내부의 시스템 남용으로 인해서 발생할 수 있지만 외부인의 침입에 의해 발생할 수 있으며, 알려져 있는 침입뿐만 아니라 알려지지 않은 침입도 탐지할 수 있다. 하지만 이상침입은 오용침입보다 탐지가 어렵다. 오용침입과 달리 일정한 패턴이 있는 것이 아니기 때문에 많은 감사 자료를 분석하여 판단하여야한다.

이상탐지의 가장 대표적인 방법인 통계적 이상탐지(statistical anomaly detection)방법은 어떤 특정한 사건의 발생빈도를 관측하여 평소보다 특이하게 높거나 낮으면 이상 침입으로 판단한다. 통계적 방식은 단순히 빈도를 관측하는 것으로서 여러 사건들이 복합적인 관계를 가지고 발생하는 경우에는 이러한 복합적인 관계를 감안하기 힘들다. 또한 통계적 방식은 정상적인 사건임에도 불구하고 이전에 관측되지 않았던 사건을 침입으로 판단하는 오탐(false alarm)의 가능성이 크다. 이를 극복하는 방법으로 명세 기반 이상탐지 방법이 [18, 19]에서 제안되었다.

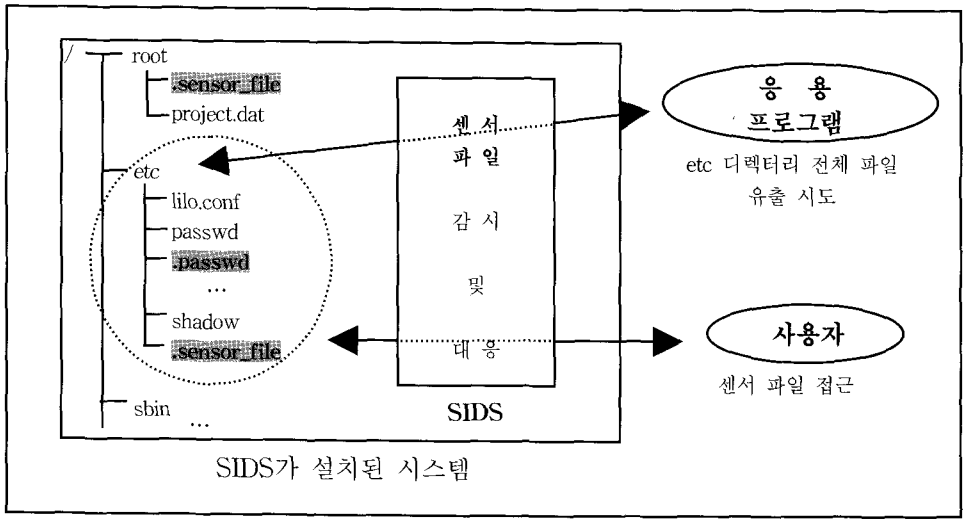
본 연구는 센서 객체를 기반으로 침입을 탐지한다는 접근방식 측면에서 기존의 오용침입 탐지나 이상침입 탐지 방법과는 차별된다. 즉, 다양한 침입 사고의 패턴들을 사전에 수집하고 이러한 패턴의 상황이 발생하는지를 감시하는 오용침입 탐지 방법과는 달리 본 연구에서 제안한 방법은 알려지지 않은 공격에 대한 탐지도 가능하다. 한편, 본 연구에서 제안한 방법은 이상침입 탐지 방법에서 야기될 수 있는 오탐의 문제도 발생하지 않는다. 이는 센서 객체를 접근하는 시도들은 정해진 모델을 벗어나는 비정상행위임이 분명하기 때문이다.

침입탐지시스템은 탐지에 사용되는 데이터 소스에 따라 호스트 기반의 침입탐지시스템과 네트워크 기반의 침입탐지시스템으로 구분 할 수 있다[8]. 호스트 기반의 침입탐지시스템은 시스템 로그 정보와 특정 행위에 대한 감사 자료 분석 등 시스템 내부에서 생성되는 정보에 대한 분석을 통하여 침입을 탐지하는 기법이다. 내부 사용자나 외부 침입자의 불법적인 시스템 사용이나 시스템 데이터의 불법적인 변경으로부터 시스템을 안전하게 보호하는 기능을 수행하며, 각 호스트를 개별적으로 모니터링을 할 수 있기 때문에 보다 정확한 탐지 능력을 발휘할 수 있다. 하지만 호스트 기반의 침입탐지시스템은 설치되는 호스트의 성능 저하를 야기하며, 네트워크를 통한 공격을 탐지하는데 어려움이 있다.

네트워크 기반의 침입탐지시스템은 네트워크 상의 패킷 헤더 및 데이터를 분석하거나 패킷 트래픽량 등을 분석하여 침입 유무를 판단한다. 즉, 네트워크 기반 침입탐지시스템은 시



(그림 1) SIDS 구성도



(그림 2) 센서 파일

스템의 감사 자료가 아닌 네트워크 패킷을 사용해 침입을 탐지하는 시스템으로서, 네트워크를 통과하는 패킷 정보를 분석해서 공격을 탐지하고 관리자에게 보고 및 실시간 대응을 한다. 네트워크 기반의 침입탐지시스템의 경우 하나의 탐지기를 이용하여 여러 호스트를 감시할 수 있지만 암호화된 패킷을 분석하지 못하며, 침입시도 판단의 근거를 시간간격과 전송된 패킷 수에 대한 고정된 임계 값에 두고 있을 경우 이를 회피하는 공격시도에 취약하다는 단점이 있다[3]. <표 1>은 호스트 기반의 침입탐지시스템과 네트워크 기반의 침입탐지시스템의 차이점을 정리한 것이다.

〈표 1〉 호스트 기반과 네트워크 기반 침입탐지시스템의 비교

IDS / 비 교	호스트 기반 IDS	네트워크 기반 IDS
운영체제와의 관계	종속적	독립적
침입탐지 시기	비교적 늦음	상대적으로 빠름
침입흔적 삭제	가능	불가능
서비스 거부공격 탐지	제한적	가능
암호화된 침입	탐지 가능	탐지 불가능
공격루트 탐지 범위	시스템과 네트워크	네트워크에 제한적

3. 시스템 설계

본 장에서는 기존의 침입탐지시스템의 단점을 보완할 수 있는 "센서" 기반의 침입탐지시스템(Sensor based Intrusion Detection System)의 구성 및 기능에 대해 설명한다.

3.1 SIDS 구성

SIDS는 네트워크 기반의 침입탐지시스템과 호스트 기반의 침입탐지시스템의 기능이 결합된 시스템이다. SIDS에서 중요한 소프트웨어 객체로 "센서"를 사용하여 침입을 탐지하고 시스템을 보호하게 된다.

"센서"는 주요 디렉터리를 보호하는 "센서 파일" 객체와 중요 파일을 보호하는 "센서 데이터" 객체로 나뉘게 된다. "센서" 객체는 침입자를 잡기 위한 일종의 덫으로 관리자를 제외한 누군가에 의해 센서객체가 건드려지게 되면 침입 또는 이상행위라고 판단되어, 적절한 대응을 하게 된다. (그림 1)은 SIDS의 전체적인 구성도를 보여 준다.

3.2 센서 파일

"센서 파일"은 주요 디렉터리에 설치되어 디렉터리내의 파

일을 보호하는 것을 목적으로 사용된다. 이 파일은 시스템 관리자가 설치하는 파일로 일반적인 응용프로그램이 실행될 때 전혀 접근할 필요가 없는 파일이다. 따라서 만일 응용프로그램이 수행 중에 "센서 파일"을 접근한다면 그 프로그램 내부에 수상한 기능이 있는 것으로 간주할 수 있다.

(그림 2)는 "센서 파일"의 사용 예를 보여주고 있다. 현재 시스템에는 /root 디렉터리와 /etc 디렉터리에 .sensor_file, .passwd라는 이름의 "센서 파일"이 설치되어 있다. 이 파일들은 정상적인 프로그램처럼 보이지만 내부적으로 특수한 목적(파일 삭제 또는 유출)을 위한 기능을 실행하는 트로이목마와 같은 악성 프로그램의 행위를 탐지할 때 이용된다. 예를 들어 트로이목마 프로그램이 사용자의 요청에 대한 정상적인 기능을 제공하면서 내부적으로 /etc 디렉터리내의 모든 파일을 외부로 유출하려고 한다면 "센서 파일(.sensor_file과 .passwd)" 역시 접근하게 되고 이때 SIDS에 의해 탐지되게 된다.

침입자나 트로이목마와 같은 프로그램을 탐지하는 능력은 "센서 파일"이 얼마나 효율적으로 중요 디렉터리에 설치되어 있는가에 의해 결정된다. 만일 "센서 파일"이 침입자의 관심 밖의 디렉터리에 설치되어 있다면 (예를 들어 임시 디렉터리(/temp)), "센서 파일"의 탐지 효과는 매우 적을 것이다. 따라서 본 연구에서는 보호하려는 호스트 시스템의 각 디렉터리를 중요도에 따라 구분하고, 설치되는 "센서 파일"의 수와 위치를 체계적으로 관리할 수 있도록 하였다. <표 2>는 본 연구에서 사용한 리눅스 시스템에서 디렉터리의 중요도와 "센서 파일" 설치 위치를 정리한 것이다.

〈표 2〉 디렉터리의 중요도에 따른 센서 파일의 설치

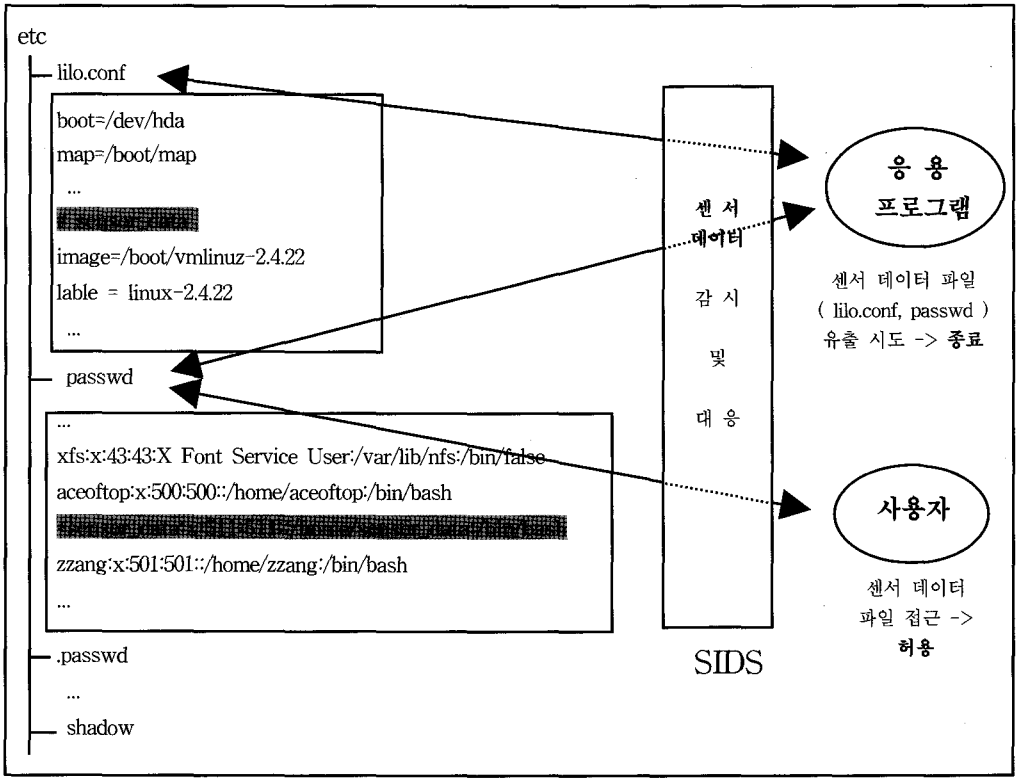
형 태	중 요 도	설 치
시스템 디렉터리	2	/boot, /etc, /root, /sbin, /dev, ...
소스 디렉터리	2	/usr/aceoftop/source, /usr/project, ...
홈 디렉터리	1	/home, /home/{aceoftop, cychang, ...}

3.3 센서 데이터

"센서 데이터"는 중요 파일 내부에 설치가 되어, 파일 자체를 보호하는 목적으로 사용된다. 로컬 시스템에서 응용 프로그램 및 사용자가 "센서 데이터"를 포함한 파일을 접근하는 것은 가능하다. 하지만 중요한 파일을 접근하는 것이므로 SIDS에서 프로그램 및 사용자의 접근을 로그로 남기게 된다. 한편, 응용 프로그램이 "센서 데이터"를 포함한 파일을 네트워크를 통해 외부로 유출하려고 한다면 해당 프로그램을 수상한 프로그램으로 가정하고 SIDS에서 탐지 및 대응을 하게 된다.

(그림 3)은 "센서 데이터"의 사용 예를 도시한 것이다. 응용 프로그램이 "센서 데이터"를 포함한 파일을 로컬 시스템에서 사용하면 로그만 남기며, 외부로 유출하려고 하면 종료시키게 된다.

"센서 데이터"를 설치하는데 있어 고려해야 할 것은 파일에 "센서 데이터"가 포함됨으로써 해당 파일을 사용하는 정상적인 응용 프로그램 및 사용자에게 영향을 미치지 않아야 한다는 것이다. 또한 파일을 읽을 때마다 "센서 데이터"가 포함되었는지 검사하기 때문에 오버헤드를 줄이기 위해선 파일 내부에 "센서 데이터"의 설치 위치가 중요하게 된다. 설치 위



(그림 3) 센서 데이터

```
//
// SIDS를 위한 가로채기용 시스템 호출 함수 구현 예
//
SIDS_sys_open()                    // open 시스템 호출에 대한 가로채기 함수
{
   ...
}

int init_module(void)              // 모듈 시작 함수
{
    // 시스템 호출 테이블의 기존 함수 시작 주소를 임시 변수에 저장
    orignal_sys_open=sys_call_table[__NR_open];
    // 시스템 호출 테이블에 가로채기 함수 시작 주소를 등록
    sys_call_table[__NR_open] = SIDS_sys_open;
    ...
}

void cleanup_module(void)          // 모듈 종료 함수
{
    // 시스템 호출 테이블에 기존 함수 시작 주소를 등록
    sys_call_table[__NR_open] = orignal_sys_open;
    ...
}
```

(그림 4) 모듈의 기본적인 형태와 시스템 호출 가로채기 예

치에 따른 성능 분석은 5장에서 자세히 설명된다.

3.4 보호 기법

"센서 파일"에 대한 접근은 침입자 및 수상한 사용자의 행동으로 가정하므로 "센서 파일"이 접근된 순간 해당 프로세스에 대한 로그를 기록하고 종료시킨다. 또한 그 순간부터 시스템의 모든 파일에 대한 삭제를 금지 시킨다. 이를 통해 침입자의 고의적인 자료 삭제를 막게 되며, SIDS를 설치한 관리자만이 모든 삭제 금지 설정을 해제 할 수 있다. 설정이 해제되면 그 후 정상적인 삭제가 가능하게 된다.

"센서 데이터"가 포함된 파일을 접근했을 경우 해당 프로세스 및 사용자의 정보를 기록한다. 로컬 시스템에서는 자유롭게 사용 가능하므로 접근을 했다고 프로세스를 강제로 종료시키지는 않는다. 만일 이 파일을 외부로 유출하려고 한다면 SIDS에서 탐지하여 종료시키게 되며, 현재 접속한 사용자의 계정으로는 더 이상 접속이 되지 않도록 하여 계속되는 접근 시도 및 공격을 방지하게 된다. 이 역시 SIDS를 설치한 관리자만이 설정을 해제 수 있다.

"센서 파일" 및 "센서 데이터"의 이름이 고정되어 있는 경우 악성 프로그램 및 사용자가 "센서"에 대한 정보를 알고 있다면, "센서"를 피할 수 있다. 따라서 본 연구진은 "센서" 객체 설치 시 임의적으로 이름을 바꾸게 하여 시스템의 보안을 높일 수 있게 하였다.

4. SIDS 설계 및 구현

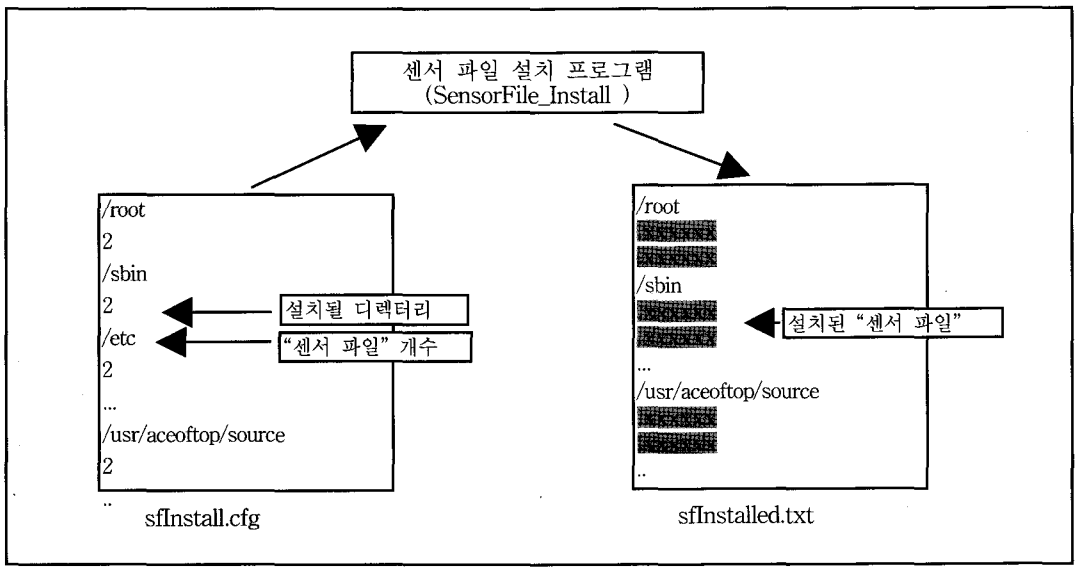
본 장에서는 실제 리눅스 시스템에서 SIDS을 구현한 내용을 설명한다. SIDS는 IBM 호환 PC 펜티엄 III, X86 프로세서용 리눅스 커널 버전 2.4.22 상에서 시스템 호출(system call) 가로채기(Hooking) 방법을 사용하여 구현되었다.

4.1 SIDS 구현

단일 커널(monolithic kernel) 구조인 리눅스의 경우 모든 시스템 기능이 단일한 주소공간의 커널에 밀집되어 있다. 따라서 커널에 새로운 기능을 추가할 경우 커널 전체 코드를 새로 컴파일을 하고 시스템을 재 시작해야 한다. 이것은 상당한 시간과 노력이 필요한 작업으로, 리눅스는 이러한 단점을 극복하기 위해 적재 가능한 커널 모듈(Loadable Kernel Module 또는 Module)이라는 방법을 제공하여 새로운 기능의 개발 및 추가가 손쉽도록 했다[5, 10].

모듈은 커널에서 필요할 때 로드(load)와 언로드(unload)가 가능한 객체로 시스템을 재부팅 할 필요 없이 커널의 기능을 확장 할 수 있는 방법이다[10]. 하나의 잘못된 모듈로 인해 커널이 정지할 수 있다는 단점은 있지만, 신중히 작성하면 많은 장점을 얻을 수 있다. 모듈은 로드 된 후에는 커널 코드와 동등한 권한으로 실행되기 때문에 성능의 저하 없이 효율적으로 작동하며 특권 명령을 수행할 수 있다. 또한 사용자가 특정한 기능이 필요할 때마다 로드를 시키고 언로드를 할 수 있기 때문에 메모리를 효율적으로 사용 할 수 있다.

SIDS는 모듈을 이용해 구현되었으며, 기존 시스템 호출의 가로채기(Hooking)를 통해 "센서 파일"과 "센서 데이터"를 관리한다. 모듈의 기본 인터페이스는 init_module()과 cleanup_module()이며(최근 리눅스 커널에서는 module_init()과 module_exit()로 인터페이스 이름이 변경되었지만 기능은 유사하다), 이를 이용한 sys_open() 시스템 호출 가로채기 예가 (그림 4)에 도시되어 있다. 모듈을 로드하는 리눅스 명령어인 insmod는 모듈의 init_module() 함수를 실행시키는데, 이때 커널의 시스템 호출 테이블(sys_call_table)을 수정하여 본래



(그림 5) "센서 파일" 설치

```
SIDS_sys_open()  // open 시스템 호출 가로채기 함수
{
        // 센서 파일의 존재 여부 파악 (SIDS를 위해 추가된 코드)
        while( 마지막 "센서 파일" 까지 ) {
                if( "센서 파일" 이면 ) {
                        로그 기록 및 대응(종료)
                } else
                        break;
        }

        // 기존의 open 시스템 호출로 제어를 넘김
        original_sys_open();
}
```

(그림 6) SIDS_sys_open() 의사코드

시스템 호출 함수(sys_open)의 주소를 SIDS 함수(SIDS_sys_open)로 바꾸게 된다. 반면 모듈을 언로드하는 rmmod 명령은 cleanup_module() 함수를 실행하며, 원래의 시스템 상태로 돌아가게 한다[1, 6, 10, 11].

SIDS를 구현하기 위해 기존 시스템 호출 함수 중 sys_open(), sys_read(), sys_close(), sys_socketcall(), sys_fork(), sys_unlink()에 대한 가로채기(hooking) 함수를 구현하였다. 가로채기 함수들의 이름은 리눅스 커널의 전통(convention)에 따라 "SIDS_"라는 접두어를 붙였다. SIDS_sys_open()과 SIDS_sys_close()는 "센서 파일"에 대한 접근을 탐지할 때 이용된다. SIDS_sys_read()는 "센서 데이터"에 대한 로컬 접근을 탐지할 때 이용되며, SIDS_sys_socketcall()는 "센서 데이터"에 대한 원격 접근을 탐지할 때 이용된다. 한편 SIDS_sys_fork()과 SIDS_sys_unlink()는 보호 기법을 적용할 때 이용된다. 본 논문에서는 "센서 파일" 및 "센서 데이터"의 빠른 검색을 위해 Boyer_Moore 알고리즘을 사용하였다[4, 9]. 한편 센서 객체에 대한 자동 설치 및 제거를 위하여 SensorFile_Install과 SensorData_Install 이라는 이름의 응용 프로그램을 작성하였다.

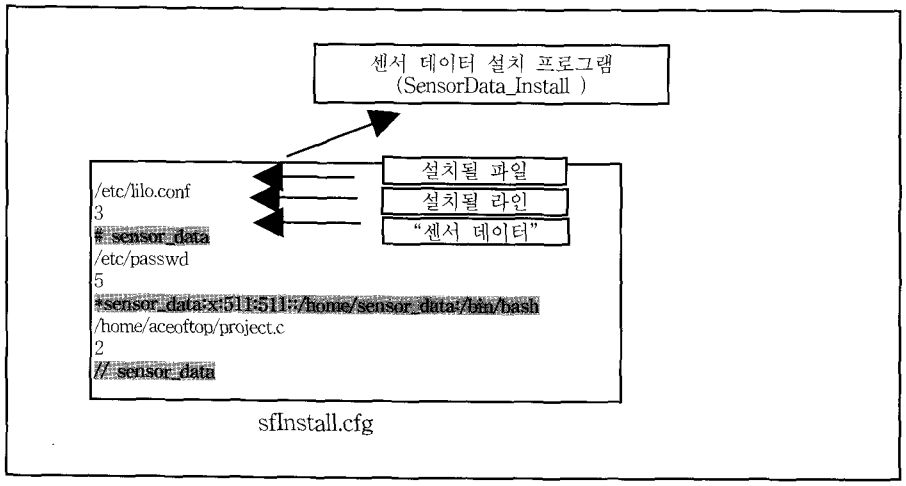
4.2 센서 파일의 설치 및 관리

"센서 파일"은 SensorFile_Install 프로그램에 의해 설치된다. 이 프로그램은 "센서 파일"이 설치 될 디렉터리와 개수를 포함하고 있는 환경파일 sfInstall.cfg를 이용하여 설치를 하게 되고 "센서 파일"의 이름을 기록한 sfInstalled.txt 파일을 생성한다. 또한 sfInstalled.txt파일은 "센서 파일"을 제거하는데 사용된다. (그림 5)는 "센서 파일"의 설치 방법을 보여준다.

SIDS가 실행되면 시스템 호출 함수 sys_open()이 호출되었을 때 sys_open() 함수 대신 SIDS_sys_open() 함수가 호출되어 서비스를 하게 된다. 모든 파일의 open에 대해서 "센서 파일"인지 검사를 하게 되며, "센서 파일"이면 로그를 기록하고 현재 프로세스를 종료시키게 된다. "센서 파일"이 아닌 일반적인 파일에 대한 open의 경우는 정상적으로 실행되게 된다. SIDS_sys_open()함수의 의사코드(pseudo code)는 (그림 6)에 나타나 있다.

4.3 센서 데이터의 설치 및 관리

"센서 데이터"는 설치 될 파일 이름과 설치될 "센서 데이터"를 포함하고 있는 환경파일인 sdInstall.cfg를 이용하여



(그림 7) "센서 데이터" 설치

```
SIDS_sys_read()  // read 시스템 호출 가로채기 함수
{
        // 기존의 read 시스템 호출 수행
        original_sys_read();

        // 센서 데이터의 존재 여부 파악 (SIDS를 위해 추가된 코드)
        while( 마지막 "센서 데이터" 까지 ) {
                if( "센서 데이터"가 존재하면) {
                        로그 기록
                }else
                        break;
        }
}
```

(그림 8) SIDS_sys_read() 의사코드

SensorData_Install 프로그램이 설치를 하게 된다. 설치된 "센서 데이터"를 제거할 경우 같은 환경파일인 sdInstall.cfg를 이용한다. (그림 7)은 "센서 데이터"의 설치 방법을 보여준다.

lilo.conf 파일에서는 '#'뒤의 문자는 주석으로 인식된다. 그러므로 '#'뒤에 "센서 데이터"를 설치함으로서 lilo.conf 파일을 이용하는 응용 프로그램에 영향을 주지 않게 된다. C언어의 소스파일일 경우는 '//'로 시작되는 라인은 주석으로 인식함으로 (그림 7)과 같이 '//'뒤에 "센서 데이터"를 설치한다. 사용자의 ID와 홈 디렉터리 등의 정보를 가지고 있는 passwd 파일의 경우는 ID 앞에 '*'이 있을 경우 그 사용자는 사용하지 않는 것으로 간주한다. 그러므로 '*'뒤에 "센서 데이터"를 설치하게 된다. 이렇게 설치된 "센서 데이터"는 lilo.conf, C 언어 소스파일, passwd 파일을 사용하는 응용 프로그램에는 영향을 미치지는 않으며, SIDS에서 탐지 및 대응 목적으로 이용되게 된다.

SensorData_Install 프로그램에 의해 설치된 "센서 파일"을 포함하고 있는 파일의 접근은 SIDS_sys_read() 함수에서 탐지가 이루어지며, 이 함수에 대한 의사 코드가 (그림 8)에 기술되어 있다. "센서 데이터"를 포함한 파일은 로컬시스템에서는 정상적으로 읽을 수 있는 파일이므로 접근한 사용자에 대한 로그만을 기록하며, 종료시키지는 않는다.

네트워크 관련함수인 sys_socket(), sys_connect(), sys_accepts(), sys_send(), sys_recv() 등은 sys_socketcall() 함수 내부에서 호출되며, 따라서 본 논문에서는 "센서 데이터"의 외부 유출 여부를 SIDS_sys_socketcall()에서 감시한다. (그림 9)는 SIDS_sys_socketcall() 함수 내용을 의사 코드로 기술한 것이다. 이 함수에서는 "센서 데이터"의 존재 유무를 검사한 후, "센서 데이터"가 포함된 파일을 전송하려고 하면 금지시킨다.

4.4 보호 기법의 구현

리눅스 시스템의 경우 파일의 삭제는 sys_unlink() 함수에서 실행이 된다. 침입자로부터의 자료의 삭제를 막기 위해 (그림 10)처럼 SIDS는 "센서 파일"이 접근되었거나 "센서 데이터"가 포함된 파일이 네트워크를 통해 유출되려고 했다면, 그 순간부터 모든 파일의 삭제를 금지시켜 자료의 삭제로 인한 피해를 줄이도록 한다.

외부에서 접속한 사용자가 "센서 파일"을 접근하거나 "센서 데이터"의 유출을 시도했다면, 그 사용자는 더 이상의 접속하지 못하도록 한다. 이 작업은 SIDS_sys_fork()에서 수행되며, 의사 코드가 (그림 11)에 기술되어 있다. 구체적으로 fork() 함수를 요청하는 사용자에 대해 "센서 파일"을 접근했거나, "센서 데이터"가 포함된 파일을 외부로 유출하려고 했

```
SIDS_sys_socketcall( ) // socket 시스템 호출 가로채기 함수
{
        // 센서 데이터의 존재 여부 파악 (SIDS를 위해 추가된 코드)
        if ("센서 데이터"가 포함되어 있으면) {
                네트워크를 통한 전송 금지
        } else {
                // 기존의 socket 시스템 호출 수행
                original_sys_socketcall()
        }
}
```

(그림 9) SIDS_sys_socketcall( ) 의사코드

```
SIDS_sys_unlink( ) // SIDS 보호 함수: 삭제 제어
{
        // 센서 객체에 대한 접근 시도가 있으면 (SIDS를 위해 추가된 코드)
        if("센서 파일" 접근 || "센서 데이터" 포함된 파일 유출 시도 ) {
                모든 파일의 삭제 금지
        } else {
                // 기존의 unlink 시스템 호출 수행
                original_sys_unlink();
        }
}
```

(그림 10) SIDS_sys_unlink( ) 의사코드

```
SIDS_sys_fork( ) // SIDS 보호 함수: 태스크 생성 제어
{
        // 센서 객체에 대한 접근 시도가 있으면 (SIDS를 위해 추가된 코드)
        if("센서 파일"접근한 사용자 || "센서 데이터" 유출하려한 사용자) {
                새로운 프로세스 생성 금지
        } else {
                // 기존의 fork 시스템 호출 수행
                original_sys_fork();
        }
}
```

(그림 11) SIDS_sys_fork( ) 의사코드

다면, 정상적인 fork()가 실행되지 않도록 하여, "센서" 객체를 접근한 사용자는 더 이상 시스템에 접근 및 일반적인 명령도 사용할 수 없도록 하였다.

5. 실험 결과

본 장에서는 SIDS의 탐지 결과와 성능 상에 부하 측정 결과를 기술한다.

5.1 센서 파일 접근

"센서 파일"을 접근했을 때 SIDS는 실행된 명령, "센서 파일" 절대 경로, 프로세스 식별자(PID), 사용자 식별자, 접근 시간 등을 기록한다. "센서 파일"은 프로그램 및 사용자가 접근할 필요가 없는 파일이므로 루트권한의 사용자라도 접근을 했다면 로그를 기록하고 해당 프로세스를 종료시키게 된다.

(그림 12)는 "센서 파일"을 접근했을 때 기록되는 로그 파일인 SIDS_open.log의 내용을 보여주고 있다. 로그로부터 cat, vi, wc 응용이 센서 파일을 접근했으며, cat의 경우 프로세스 식별자는 686, 사용자 식별자는 0 (결국 root 사용자), 접근 시간은 11시 34분 7초임을 파악할 수 있다.

```
[root@oslab01 /root]# cat SIDS_open.log
[cat] accessed /etc/.sensor3 pid = 686 uid = 0 time = 2003/12/04
thu 11:34:07
[vi] accessed /root/.sensor10 pid = 687 uid = 0 time = 2003/12/04
 thu 11:34:17
[wc] accessed /home/aceoftop/.sensor5 pid = 688 uid = 0 time = 20
03/12/04 thu 11:34:32
[root@oslab01 /root]#
[영어][완성][두벌식]
```

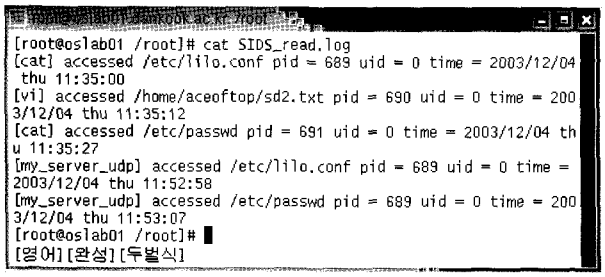
(그림 12) "센서 파일" 로그 SIDS_open.log

5.2 센서 데이터 접근

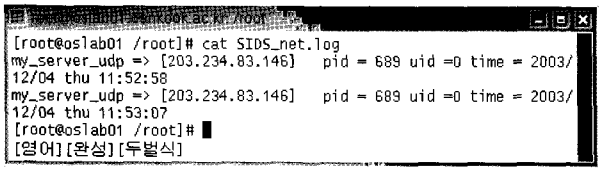
"센서 데이터"가 포함된 파일은 로컬 시스템에서만 접근 및 활용가능하며 외부로 유출되어서는 안 되는 중요한 파일이므로 접근한 모든 프로세스에 대한 정보를 기록한다. (그림 13)은 "센서 데이터"가 포함된 파일을 로컬 시스템에서 접근한 응용 프로그램들의 프로세스 식별자, 사용자 식별자, 접근

시간 등에 대한 로그 기록을 보여주고 있다.

"센서 데이터"가 포함된 파일을 유출할 경우 SIDS의 탐지 능력을 확인하기 위해 UDP를 사용하여 파일을 전송하는 my_server_udp 프로그램을 작성하여 실험해 보았다. SIDS가 기록한 (그림 14)와 (그림 13)의 로그 파일로부터 my_server_udp 라는 프로그램이 "센서 데이터"가 포함된 lilo.conf 파일과 passwd 파일을 외부로 유출하려고 했다는 것을 알 수 있다.



(그림 13) "센서 데이터" 로그 SIDS_read.log



(그림 14) "센서 데이터" 로그 SIDS_net.log

5.3 성능 분석

제안된 SIDS는 주요 디렉터리나 파일을 접근할 때 마다 "센서 파일" 또는 "센서 데이터"에 대한 접근 인지를 검사하는 추가적인 연산을 야기한다. 즉 기존시스템에 비해 파일을 열 때마다 그 파일이 "센서 파일"인지, 파일의 내용을 읽을 때마다 "센서 데이터"가 읽혀졌는지, 네트워크 패킷 내에 "센서 데이터"가 포함되었는지 등을 조사하기 위해 추가적인 시간이 필요하게 된다.

"센서 파일"의 경우는 파일의 이름을 검사하는 것이므로 설치된 "센서 파일"의 수가 시스템의 성능에 영향을 미치게 된다. 본 연구진은 /root, /etc, /var 등의 주요 시스템 디렉터리와 사용자 디렉터리에 10개에서 160개의 "센서 파일"을 설치하고 이에 따른 일반적인 파일에 대한 열기 시간과, "센서 파일"에 대한 열기 시간을 측정해 보았다. <표 3>은 실험 결과를 정리한 것이다.

〈표 3〉 "센서 파일"의 개수에 따른 성능 변화

시간 (μs) \ 센서 파일 수(개)		10	20	40	80	160
SIDS 실행	일반 파일 열기 시간(μs)	5	5	7	16	27
	센서 파일 열기 시간(μs)	23	24	33	57	60
기존 시스템	파일 접근 시간(μs)	4	4	4	4	4

기존 시스템에서 파일 열기 시간은 "센서 파일"의 개수와 독립적으로 항상 4(μs)이다. 반면 SIDS가 실행되는 환경에서는 "센서 파일"의 개수에 따라 파일의 열기 시간이 증가한다. 160개의 "센서 파일"이 설치되었을 경우 일반적인 파일 접근에 대해서는 27(μs)정도의 시간이 걸리는 것을 측정할 수 있었다. 이는 open() 함수에 요청된 파일 이름이 "센서 파일"인지 여부를 비교하기 위한 부하가 추가되었기 때문이다. 한편 "센서 파일" 접근에 대해서는 로그 기록 및 대응 시간이 추가되어 60(μs)정도 시간이 걸리는 것을 측정할 수 있었다.

"센서 데이터"는 파일 내의 존재 여부를 검사하는 것으로 "센서 데이터"를 포함하고 있는 파일의 크기와 설치된 위치에 따라 성능에 영향을 미치게 된다. 따라서 <표 4>와 같이 500Byte에서 1M 사이의 파일에서 첫 줄과 마지막 줄에 "센서 데이터"를 각각 설치하여 시간을 측정해 보았다.

〈표 4〉 "센서 데이터"가 포함된 파일의 크기에 따른 성능 변화

시간(μs) \ 파일 크기		500Byte	1K	10K	100K	1M
SIDS 실행	첫 줄에 센서 데이터 설치 시 read()함수 완료 시간(μs)	11	16	92	877	11,433
	마지막 줄에 센서 데이터 설치 시 read()함수 완료 시간 (μs)	11	17	107	1,010	12,851
기존 시스템	read()함수 완료 시간 (μs)	6	11	87	841	11,214

"센서 데이터"가 파일의 앞쪽에 설치되었을 경우, 존재 유무가 확인되었다면 나머지 부분은 검사할 필요가 없으므로 가장 적은 성능저하를 나타낸다. 반면 파일의 크기가 1M이며 마지막 줄에 센서 데이터가 있는 경우에는 "센서 파일"의 존재유무를 검사하기 위한 추가적인 계산 시간은 1.64ms인 것으로 측정되었다. 리눅스 환경에서 시스템 호출의 모니터링 기반 접근 제어를 구현한 관련 연구 [20]에 따르면, 실험 결과 open 함수의 경우 3ms, null I/O의 경우 0.48ms의 부하가 접근 제어를 위해 추가됨을 발표하였다. 비록 [20]의 구현 내용 및 환경이 본 연구의 구현 내용 및 환경이 달라 직접적인 비교를 할 수는 없지만, 본 연구도 밀리 초 단위의 적은 부하로 침임 탐지가 가능함을 성능 분석 결과 알 수 있었다.

6. 결 론

본 논문에서는 센서기반 침입탐지시스템을 설계하고 구현하였다. 사실, 본 연구에서 이용한 센서기반 시스템 보호의 개념은 이미 기존 연구 [12, 13]에서 제안되었으며, 본 논문은 이러한 개념을 "센서 파일"과 "센서 데이터"로 구체화하고 실제 상용 시스템에 적절하게 설계 및 구현하여 효과를 검증하고 시간 부하를 정량적으로 측정한 것이다.

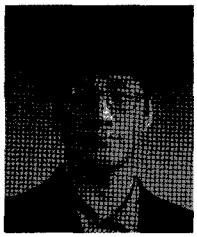
구현된 시스템은 다음과 같은 특징을 갖는다. 첫째, 주요 디렉터리 및 파일의 접근을 감시하여 트로이 목마 등의 공격 도구에 의해 중요한 데이터가 임의로 접근되거나 외부로 유출되는 것을 방지할 수 있다. 둘째, 침입 패턴이 알려진 공격뿐만 아니라 새로운 유형의 침입 시도에 대해서도 탐지가 가

능하다는 특징과 오탐(false alarm)이 발생할 가능성이 없다는 특징이 있다. 특히 Security에서 중요한 척도 중에 하나인 비도 측면에서 보면, 본 논문에서 제안한 방법은 침입접근 패턴이나 비정상적인 행위를 사전에 수집하거나 관리자가 명세하고 이를 기반으로 침입 여부를 판별하는 것이 아니라 센서 객체에 대한 접근 여부를 통해 침입 여부를 판별하는 것이기 때문에, 센서 객체가 설치된 디렉터리나 파일에 대한 침입은 모두 발견할 수 있다. 셋째, SIDS를 리눅스 운영체제상에서 LKM 방식으로 구현함으로서 커널을 직접 컴파일 해야 하는 어려움과 시스템을 재부팅 할 필요 없이 간단히 설치가 가능하도록 하였다.

향후 센서 객체의 유형과 디렉터리 위치에 따른 시스템 보안의 탐지 범위를 분류하는 연구를 진행할 계획이다. 또한 센서 객체가 접근되었을 때 커널 수준에서의 대응뿐만 아니라 시스템 프로그램과도 연계하여 전체적으로 시스템의 작동 방식이 변화되는 방법 및 입력의 변화에 따른 성능상의 영향, 그리고 제안된 시스템의 비도를 적절한 비도 벤치마크를 이용하여 정량적으로 분석할 계획이다.

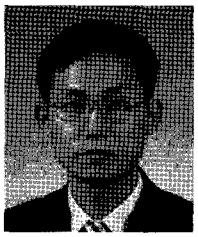
참 고 문 헌

[1] 박장수, '리눅스 커널 분석 2.4', 가메출판사, 2003. 1.
[2] 신동철, "시스템 호출 추적을 통한 리눅스 시스템에서의 침입탐지", 단국대 석사학위논문, 1999. 12.
[3] 유일선, "네트워크 취약점 검색공격에 대한 개선된 탐지시스템", 단국대 박사학위논문, 2001. 11.
[4] 이재규, 'C로 배우는 알고리즘', 세화, 2001. 2.
[5] 이호, 심마로, '리눅스 커널의 이해', 한빛미디어, 2002. 1.
[6] 이호, 'Linux Kernel Programming', http://linuxkernel.net
[7] 한국정보보호학회 편, '정보보호 관리 및 정책', 한국정보보호진흥원, 2002. 11.
[8] 한국정보보호학회 편, '차세대네트워크 보안 기술', 한국정보보호진흥원, 2002. 11.
[9] Christian Charras and Thierry Lecroq, 'Handbook of Exact String-Matching Algorithms', http://www-igm.univ-mlv.fr/~lecroq/string/string.pdf
[10] Peter Jay Salzman, 'The Linux Kernel Module Programming Guide', http://tldp.org/LDP/lkmpg/, 2003. 4.
[11] Pragmatic/THC, 'Complete Linux Loadable Kernel Modules', http://packetstormsecurity.org/docs/hack/LKM_HACKING.html, 1999. 3.
[12] Seongje Cho, Chulyean Chang, Joonmo Kim and Jongmoo Choi, "A Study on Monitoring and Protecting Computer System against Interception Threat:", LNCS(Lecture Notes in Computer Science Series) 2713, pp.96-105, June, 2003.
[13] Seongje Cho, Joonmo Kim, Hong-Geun Kim and Doohyun Kim, "A Trojan Horse Detection System Using Sensor Concept", 2002 ICOCM(International Conferrence on Optical Communications and Multimedia), pp.83-86 , November, 2002.
[14] U. Lindqvist and P. Porras, "Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST)", Proceedings of the Symposium on Security and Privacy, 1999.
[15] S. Kumar and E.H. Spafford. "A Pattern Matching Model for Misuse Intrusion Detection", Proceedings of the 17th National Computer Security Conference, 1994.
[16] C. C. Michael, Anup Ghosh, "Simple, state-based approaches to program-based anomaly detection", ACM Transactions on Information and System Security, Vol.5, Issue 3, 2002.
[17] SalvatoreStolfo, et al., "Anomaly Detection in Com- puter Security and an Application to File System Accesses", Proceedingsof 15th International Symposium of Foundations of Intelligent Systems, 2005.
[18] R. Sekar, et al., "Intrusion Detection: Specification- based anomaly detection : a new approach for detecting network intrusions", Proceedings of the 9th ACM conference on Computer and communications security, 2002.
[19] R. Sekar and P. Uppuluri, "Synthesizing Fast Intrusion Prevention/Detection Systems from High-Level Specifications", Proceedings of USENIX Security Symposium, 1999.
[20] P. Loscocco, S. Smalley, "Integrating Flexible Support for Security Policies into the Linux Operating System", Proceedings of the FREENIX Track of the USENIX Annual Technical Conference, 2001.



최 종 무

e-mail : choijm@dankook.ac.kr
1993년 서울대학교 해양학과(학사)
1995년 서울대학교 대학원 컴퓨터공학과(공학석사)
2001년 서울대학교 대학원 컴퓨터공학과(공학박사)
2001년~2003년 유비쿼스(주) 기술연구소 책임연구원
2003년~현재 단국대학교 정보컴퓨터학부 컴퓨터과학전공 조교수
2003년~현재 한국정보과학회 논문지편집위원
2003년~현재 서울대학교 컴퓨터신기술연구소 연구원
관심분야 : 시스템 소프트웨어, 내장형 시스템, 파일 시스템, 유비쿼터스 컴퓨팅 등



조 성 제

e-mail : sjcho@dankook.ac.kr
1989년 서울대학교 컴퓨터공학과(학사)
1991년 서울대학교 대학원 컴퓨터공학과(공학석사)
1996년 서울대학교 대학원 컴퓨터공학과(공학박사)
1996년~1997년 서울대학교 컴퓨터신기술연구소 연구원
2001년~2002년 미국 University of California, Irvine 객원연구원
1997년~현재 단국대학교 정보컴퓨터학부 컴퓨터과학전공 부교수
관심분야 : 컴퓨터 보안, 시스템 소프트웨어, 실시간 시스템, 임베디드 시스템 등