LxBSM: 리눅스 감사 시스템

김세환·김지홍[1]·권일혁·심원태[2]

요 약

리눅스 운영체제가 서버 운영체제-웹 서버, 메일 서버 등-로써 사용되기 위한 필수적인 요건 중의 하나는 운영체제 수준의 보안성이다. 리눅스 운영체제는 유사 상용 유닉스 운영체제에 비해 운영체제 수준의 보안성측면에서 취약하다. 특히, TCSEC(Trusted Computer System Evaluation Criteria)에서 정의한 C2 레벨 이상의 보안성을 만족시키기 위한 감사 자료 생성 기능이 대표적인 예이다. SunOS 운영체제는 SunSHIELD BSM(Basic Security Module)을 통해 C2 레벨 이상의 보안성충족을 목표하고 있다. 리눅스 운영체제 역시 C2 레벨 이상의 보안성을 만족시키기 위해선 SunSHIELD BSM 수준의 감사 자료 생성 기능이 필수적이다.

LxBSM은 SunSHIELD BSM과 동일한 수준의 감사 자료 생성을 주 목적으로 개발되었다. SunSHIELD BSM과 동일한 시점인 시스템 콜 호출 시점에서 감사 자료를 생성하며 SunSHIELD BSM에서 지원하는 거의 모든 감사 이벤트에 대해 동일한 포맷의 감사 자료를 생성한다. LxBSM은 기존의 리눅스 커널에 대해 변경 없이 적용할 수 있도록 LKM(Loadable Kernel Module) 기법으로 개발되었다. 따라서, LxBSM을 적용하기 위해 커널 이미지를 다시 생성하는 과정이 필요치 않으며 시스템의 다운타임없이 동적으로 활성화할 수 있는 장점을 지닌다. LxBSM은 침입탐지 프로세스와 같은 외부 프로세스가 생성된 감사자료를 용이하게 활용할 수 있는 인터페이스를 가지고 있다. 시스템 관리자는 이 인터페이스를 통해 LxBSM에서 제공한 수많은 기본 감사 자료들 중 보안상 의미 있는 자료들만을 처리할 수 있다. LxBSM은 커널 수준에서 동작하므로 성능 및 안정성 면에 있어서 시스템에 큰 영향을 미치지 않아야한다. LMBench와 WebBench 툴을 사용하여 LxBSM이 시스템 미치는 성능상의 영향을 측정하여 만족할만한 결과를 도출하였으며 LTP(Linux Test Project) 툴을 사용하여 안정성 측면에서도 문제를 일으키지 않음을 확인하였다.

LxBSM: Audit System for Linux

Sehwan Kim·Jihong Kim[3]·Ilhyuk Kwon·Wontae Sim[4]

1) 서울대학교 전기 컴퓨터 공학부 컴퓨터구조 및 내장형시스템 연구실
2) (주)인젠 기술 연구소
3) Computer Architecture and Embedded Systems Laboratory, School of Computer Science & Engineering, Seoul National University

ABSTRACT

One of the required features for the adoption of Linux operating system as a server operating system is the security of the operating system. Linux operating system should provide same or similar level of security compared to mainstream operating systems. Linux operating system lack of auditing functionality to satisfy the C2 level audit requirements as defined in TCSEC(Trusted Computer System Evaluation Criteria) documents, while other mainstream operating systems are in better situation than Linux operating system - Especially, SunOS-. SunOS provide SunSHIELD BSM(Basic Security Module), which is an security system for SunOS - to satisfy the C2 level requirements. Linux operating system - at least - also need to generate the same kinds of the audit trails which are provided by SunSHIELD BSM to satisfy the C2 level or higher level requirements. LxBSM was developed for the generation of the same audit trails provided by SunSHIELD BSM. LxBSM generates the audit trails at the moment of the system calls' invocation, which was developed as LKM (Loadable Kernel Module) not to require the modifications in the existing Linux kernel routines. Additionally, we ported the commercial host-based intrusion detection system to Linux platform and validated the inter-operation between LxBSM and host-based IDS system.

1. 서론

리눅스 운영체제는 다양한 영역 -산업체, 학계, 공공기관, 연구기관 등- 에서 여러 용도의 서버 운영체제 -웹 서버, 메일서버, 뉴스서버, 네임서버, 파일서버 등- 로 운용되고 있으며 확대 적용되고 있다.

리눅스 운영체제가 서버 운영체제로써 사용되기 위한 필수적인 요건 중의 하나는 운영체제 수준의 보안성이다. 리눅스 운영체제는 유사 상용 유닉스 운영체제에 비해 운영체제 수준의 보안성측면에서 취약하다. 특히, TCSEC(Trusted Computer System Evaluation Criteria)[16]에서 정의한 C2 레벨 이상의 보안성을 만족시키기 위한 감사 자료 생성 기능이 대표적인 예이다. SunOS 운영체제는 SunSHIELD BSM(Basic Security Module)[14]을 통해 C2 레벨 이상의 보안성충족을 목표하고 있다. 리눅스 운영체제 역시 C2 레벨 이상의 보안성을 만족시키기 위해선 SunSHIELD BSM 수준의 감사 자료 생성 기능이 필수적이다.

LxBSM은 SunSHIELD BSM과 동일한 수준의 감사 자료 생성을 주 목적으로 개발되었다. SunSHIELD BSM과 동일한 시점인 시스템 콜 호출 시점에서 감사 자료를 생성하며 거의 모든

---

4) Research and Development Center, Inzen Corporation

감사 이벤트에 대해 동일한 포맷의 감사 자료를 생성한다. LxBSM은 기존의 리눅스 커널에 대해 변경 없이 적용할 수 있도록 LKM(Loadable Kernel Module) 기법으로 개발되었다. 따라서, LxBSM을 적용하기 위해 커널 이미지를 다시 생성하는 과정이 필요치 않으며 시스템의 다운타임없이 동적으로 활성화할 수 있는 장점을 지닌다. LxBSM은 침입탐지 프로세스와 같은 외부 프로세스가 생성된 감사자료를 용이하게 활용할 수 있는 인터페이스를 가지고 있다. 시스템 관리자는 이 인터페이스를 통해 LxBSM에서 제공한 수많은 기본 감사 자료들 중 보안상 의미 있는 자료들만을 처리할 수 있다. LxBSM은 커널 수준에서 동작하므로 성능 및 안정성 면에 있어서 시스템에 큰 영향을 미치지 않아야한다. LMBench[5]와 WebBench[17] 툴을 사용하여 LxBSM이 시스템 미치는 성능상의 영향을 측정하여 만족할만한 결과를 도출하였으며 LTP(Linux Test Project)[15] 툴을 사용하여 안정성 측면에서도 문제를 일으키지 않음을 확인하였다.

본 논문은 다음과 같이 구성되었다. 2장에서 관련 연구 현황을 알아보고, 3장에서 LxBSM의 개요를 논의하며, 4장에서 LxBSM의 구현 내용을 소개하고, 5장에서 실험 결과, 6장에서 향후 연구 계획을 논한다.

2. 관련 연구

리눅스 운영체제에 포함되어 운용되는 디폴트 감사 시스템(syslog, klog)은 수많은 커널 루틴이나 각종 데몬 들이나 전자메일(electronic mail) 프로그램등과 같은 시스템 유틸리티들에서 생성하는 로깅(logging) 관련 감사 자료들만을 처리하고 있다[7]. 즉, 리눅스 운영체제상의 디폴트 감사 시스템은 TCSEC에서 정의한 C2 수준의 보안성을 고려하지 않은 감사 시스템이다. 이로 인해 운영체제수준에서 제공되는 감사 자료를 기반으로 동작하는 상위 레벨의 보안 응용 프로그램들-예, 호스트 기반 침입탐지시스템-은 리눅스 운영체제를 지원하지 못 하고 있는 실정이다.

감사 시스템이 존재하지 않거나 충분치 못한 감사 자료를 제공하는 시스템에 대한 직접적인 응용 프로그램 수준에서의 제어도 가능하나 불안정한 감사 자료 생성 및 성능상의 심각한 결과를 초래한다[4].

운영체제에서 기본적으로 제공하지 않는 기능들은 확장 모듈(LKM)을 통해 구현가능하다[3]. 리눅스 운영체제와 같이 운영체제수준에서 감사 자료 생성 기능이 취약한 경우 LKM기법을 통한 확장 모듈의 사용으로 운영체제수준에서의 감사 자료 생성 기능을 보완할 수 있다.

U.C. Davis의 Linux BSM[6]과 SNARE[12] 모두 LKM 기법을 활용하여 시스템 콜 인터포지션(system call interposition)을 통해 감사 자료를 생성한다는 점에선 LxBSM과 동일하다. 하지만, 두 프로젝트 모두 SunSHIELD BSM에서 명시하고 있는 감사 이벤트 중 일부만을 지원하고, 동일한 포맷의 감사 자료를 생성하지 않으며, 커널 공간에서 생성한 감사 자료를 사용자 공간의 감사 데몬에 전달하는 방식이 시그널(SIGNAL) 방식에 의존함으로써 안전한 감사 자료 전달과 성능면에서 심각한 문제가 존재한다. 또한, Linux BSM의 경우는 동적으로 로딩하는 방식이 아

닌 커널 패치를 제공하는 기법이기 때문에 이를 활성화시키기 위해선 커널 이미지의 재 생성 및 시스템 재 시작 과정을 요구하는 단점이 존재한다.

LxBSM은SunSHIELD BSM에서 명시하는 거의 모든 감사 이벤트를 동일한 생성 시점과 동일한 자료 형식으로 제공하며 감사 자료의 안정적인 전달과 성능 향상을 위해 시그널 방식에 의존하지 않고 피포(FIFO) 방식으로 구현되었다. 또한, 기존 리눅스 커널 루틴에 대한 수정이 불필요하므로 활성화를 위한 커널 이미지의 재 생성 및 시스템 재 시작 과정이 불필요하다.

3. LxBSM 개요

3.1. 설계 목적

LxBSM은 리눅스 운영체제상에서 C2 레벨을 충족시키기 위한 감사 시스템이다. 이를 위해 SunSHIELD BSM에서 명시한 모든 감사 이벤트를 동일한 포맷으로 동일한 시점에서 생성하는 것이 첫 번째 설계 목적이다. 하지만, LKM기법을 활용한 제약사항 때문에 코어(Core), 로긴페일(Login Fail) 이벤트에 대해서는 감사 자료를 생성하지 않는다.[1)]

두 번째, LxBSM은 서버 용도로 운영되는 리눅스 운영체제상의 호스트 기반 침입 탐지 시스템과 같은 보안 관련 외부 응용 프로그램이 감사 자료를 쉽게 활용할 수 있도록 설계되었다. LxBSM은 생성된 감사 자료를 활용할 수 있도록 피포(FIFO) 인터페이스를 제공하며, 외부 응용 프로그램들은 이 인터페이스를 통해 실시간으로 감사 자료를 제공받을 수 있다.

세 번째 설계 목적은 서버 시스템의 다운타임(Downtime)없이 LxBSM을 적용할 수 있도록 하는 것이다. 이를 위해 LxBSM의 핵심 모듈은 LKM 형태의 모듈로 개발되었다. LxBSM은 동적으로 시스템에 활성화 될 수 있으므로 SunSHIELD BSM을 활성화시키기 위해 필요한 시스템의 재 시작 과정을 생략할 수 있다[14].

LxBSM의 활성화로 인한 시스템 부하 와 안정성에 미치는 영향을 최소화하는 것이 네 번째 설계 목적이다. LxBSM 시스템은 응용 프로그램과 달리 커널 루틴의 일부분으로써 운영체제내의 스케줄러에 의한 스케줄링 타겟(Scheduling Target)이 되지 않기 때문에 성능 면에서의 큰 오버헤드는 시스템의 운영에 심각한 영향을 줄 수 있으며, 불안정한 동작으로 인해 시스템에 대해 심각한 장애를 유발할 수 있다.

3.2. 전체구성

(그림 1)에서 보는 바와 같이 LxBSM은 드라이버 모듈, 랩퍼 모듈, 데몬 모듈 세 부분으로 구성된다. 드라이버 모듈은 커널 모드에서 생성된 감사 자료를 처리하는 모듈이다. 랩퍼 모듈은 시스템 콜 인터포지션을 통해 시스템 콜 호출을 가로채서 감사 자료를 생성하는 모듈이다. 데몬 모듈은 감사 이벤트에 대한 필터링[2)] 및 감사 파일이 저장되는 파일 시스템의 최소 사용 가능

---

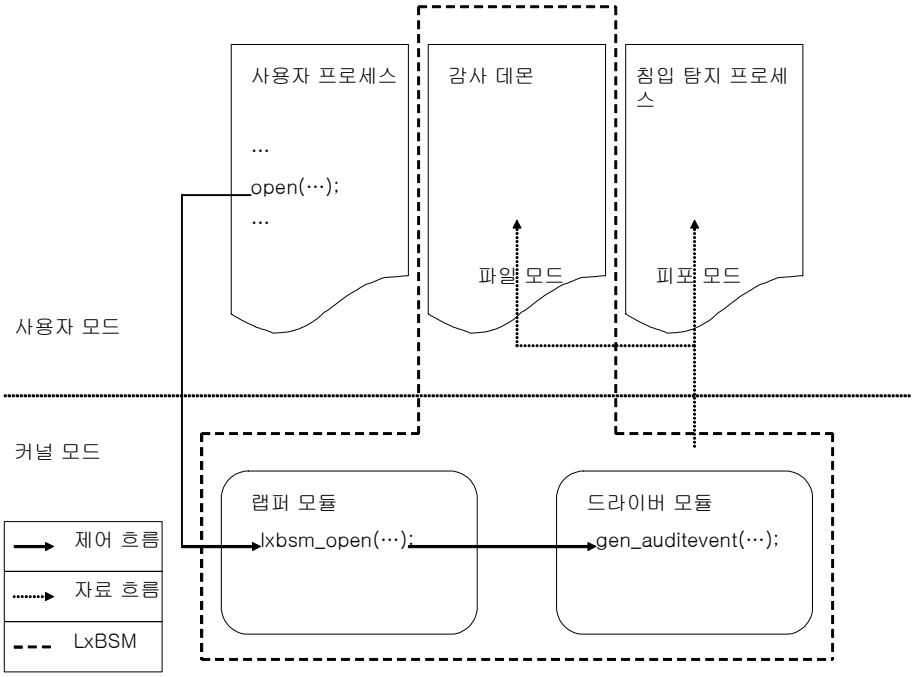
1) 코어 이벤트를 생성하기 위해 커널의 시그널 처리 루틴에 대한 수정이 필요하며 로긴 페일의 경우 로긴 프로그램에 대한 수정이 요구된다.

2) 이벤트 마스킹, 시스템 콜 성공/실패에 대한 감사 자료 생성 여부 결정등

공간 확보를 위한 설정, 그리고 감사 데몬 운용 등과 관련된 작업을 수행하는 모듈이다.

사용자 모드에서 수행되는 임의의 프로세스에서 호출한 open() 시스템 콜은 랩퍼 모듈에 구현된 lxbsm_open() 루틴으로 리다이렉션(redirection)되며, lxbsm_open() 루틴에서 open() 이벤트에 해당하는 감사 자료를 생성한 후, 드라이버 모듈에 구현된 루틴을 통해 감사 자료가 처리된다.

LxBSM은 두 가지 모드로 동작할 수 있도록 설계 되었다. 먼저, LxBSM에서 생성한 감 사 자료를 실시간으로 제공받아 침입 탐지를 수행할 수 있는 침입탐지 시스템과 연동을 고려하여 피포(FIFO) 모드로 동작 할 수 있다. 두 번째 모드에서는 LxBSM에서 생성한 감사 자료를 파일로 저장할 수 있다.

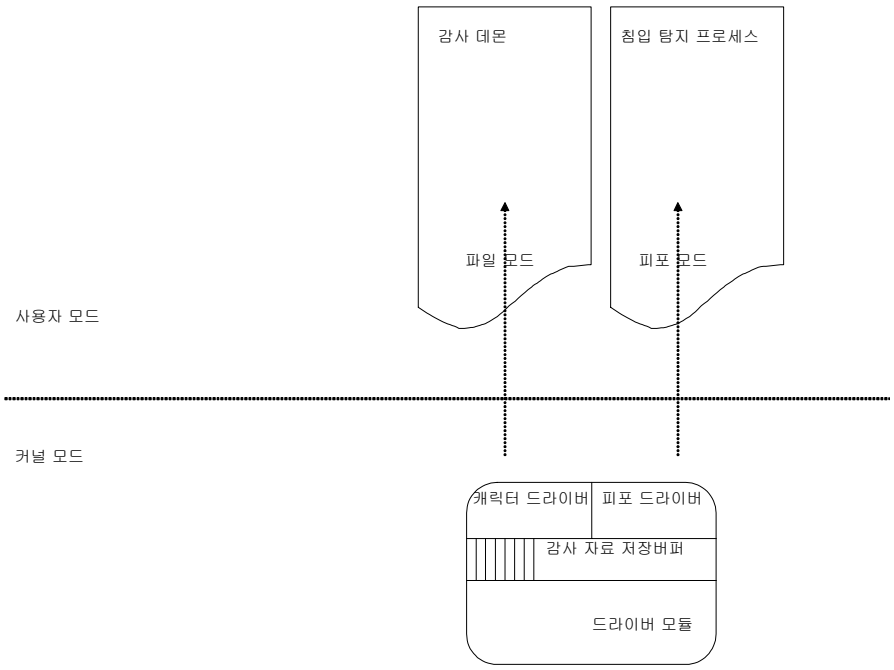


(그림 1) LxBSM 아키텍쳐

3.3. 드라이버 모듈 아키텍처

드라이버 모듈은 (그림 2)에서 보는 바와 같이 랩퍼 모듈에서 생성한 감사 자료를 임시적으로 저장하기 위한 메모리를 할당하며 피포 모드를 위한 피포 드라이버와 파일 모드를 지원하기 위한 캐릭터 디바이스 드라이버로 구성되었다.

드라이버 모듈은 랩퍼 모듈로 하여금 감사 자료를 처리하는[1] 루틴을 호출할 수 있도록 엑스포트(export)한다[2]. 피포 모드나 파일 모드 모두 일반적인 파일 I/O 인터페이스를 제공함으로써, 침입 탐지 프로세스나 감사 데몬 등에 대한 감사 자료 전달은 일반적인 파일 I/O를 통해 가능하다.



(그림 2) 드라이버 모듈 아키텍처

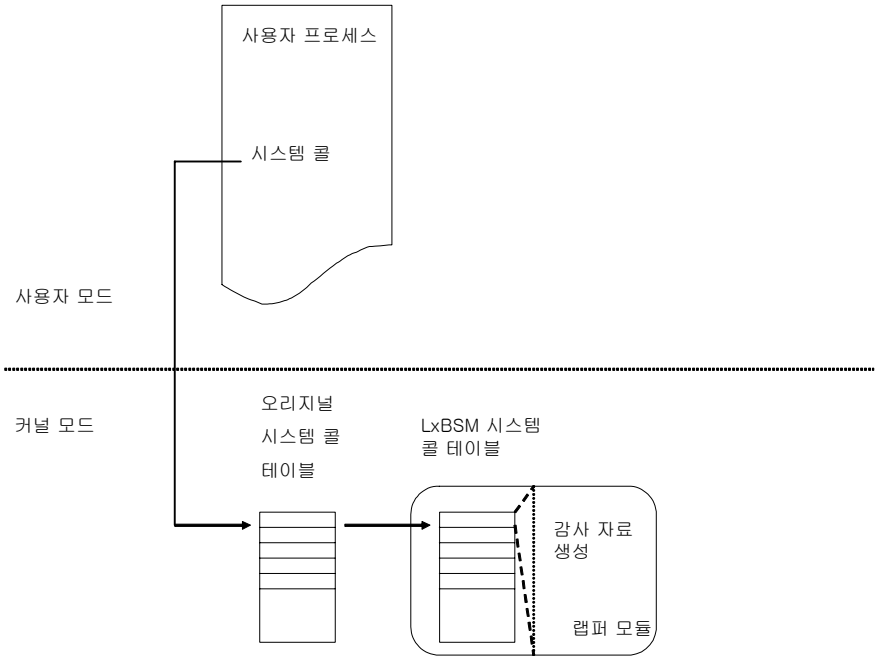
1) 피포 모드나 파일 모드로

3.4. 랩퍼 모듈 아키텍처

랩퍼 모듈은 커널에서 엑스포트한 시스템 콜 테이블의 엔트리(entry)를 우리가 원하는 랩퍼 루틴들로 교체하는 부분과 오리지널(original) 시스템 콜 호출 이전의 감사 자료 생성 부분과 호출 이후 감사 자료 생성 후 드라이버 모듈로 하여금 생성된 감사 자료를 처리하도록 요청하는 부분으로 구성된다(그림 3).

랩퍼 모듈은 LKM(Loadable Kernel Module)의 구조를 지니고 있기 때문에 커널에서 엑스포트한 시스템 콜 테이블에 접근이 가능하다.

모든 감사 이벤트는 오리지널 시스템 콜 호출의 반환값을 처리하는 리턴 토큰(return token)[14]을 생성할 필요가 있기 때문에 오리지널 시스템 콜을 랩퍼 루틴 내에서 호출하는 구조를 지닌다.



(그림 3) 랩퍼 모듈 아키텍처

3.5. 데몬 모듈 아키텍처

데몬 모듈은 위에선 언급한 바와 같이 감사 관련 설정 부분과 LxBSM이 파일 모드로 동작할 경우를 위한 처리 부분으로 구성된다.

감사 관련 설정 파일은 /etc/security/audit_* 파일들에 저장되어 있다[14]. 따라서, 데몬 모듈은 이 파일을 읽어 들인 뒤 설정값들을 드라이버 모듈에 반영시키기 위하여 드라이버 모듈에 구현된 캐릭터 디바이스 드라이버 루틴을 호출하는 구조를 지닌다.

파일 모드로 동작할 경우엔 데몬 프로세스에서 할당한 메모리 영역으로 드라이버 모듈의 버퍼에 저장된 감사 자료를 옮기기 위해 드라이버 모듈의 캐릭터 드라이버에 대한 드라이버 루틴을 호출한다.

4. LxBSM 구현

4.1. 드라이버 모듈 구현

드라이버 모듈은 LKM(Loadable Kernel Module)로 구현되었으며, 앞 절에서 언급한 바와 같이 두 개의 디바이스 드라이버로 구현되었다.

파일 모드를 위한 캐릭터 디바이스 드라이버에선 감사 데몬 프로세스 영역에 대한 mmap() 요청을 처리해 주기 위해 remap_page_range() 커널 API를 통해 구현되었다[2]. 피포 모드의 경우 시스템 피포의 사용을 고려하였지만 하나의 감사 이벤트 당 요구되는 버퍼 사이즈가 디폴트 크기는 4K 바이트보다 클 수 있다는 점[1]에서 야기될 수 있는 피포 리더와 피포 롸이터(writer)사이의 컨택스트 스위칭(context switching)을 방지하여 최적화된 성능을 보장하고 간결한 감사 자료 처리[2]를 위해 새로운 피포 드라이버를 구현하였다[9]. 또한, 랩퍼 모듈에서 생성한 감사 자료를 파일 모드나 피포 모드로 처리하기 위한 감사 자료 저장 버퍼는 성능 측면을 고려하여 get_free_pages() 커널 API(Application Programming Interface)를 통하여 물리적으로 연속된 페이지에 할당하였다[2].

4.2. 랩퍼 모듈 구현

랩퍼 모듈도 역시 LKM(Loadable Kernel Module)로 구현되었다. 랩퍼 모듈은 커널 모드에서 동작하는 루틴이기 때문에 커널에서 엑스포트된 시스템 콜 테이블에 대한 접근이 가능하며 이 테이블의 엔트리는 단순히 해당 시스템 콜의 커널 내부 루틴에 대한 함수 포인터이므로, 이에 대한 리다이렉션은 손쉽게 가능하다. open()과 같은 오리지널 시스템 콜들은 lxbsm_open()과 같이 랩퍼 모듈에 구현된 루틴으로 대체가 가능하다. LxBSM은 SunSHIELD BSM에서 명시한 거의 모든 감사 이벤트에 대한 감사 자료 생성이 가능하다.

리눅스 운영체제는 audit uid와 같이 사용자가 시스템에 로긴한 후 로긴 세션이 종료되지 않은 이상 유지되는 uid가 존재하지 않는다. setuid 시스템 콜을 통해서도 변경되지 않는 id정보를

---

1) execve 이벤트의 경우

2) 드라이버 내부 감사자료 버퍼 관리상

저장하기 위해 LxBSM에서는 audit uid를 사용자의 로긴 시점에서 설정한 후 setuid 시스템 콜에 의해서도 uid정보를 유지할 수 있도록 하는 새로운 자료 구조를 구현하였다. 이 자료 구조에 대한 관리는 운영체제 수준에서 관리하는 프로세스 ID(process ID)와 같은 메커니즘으로 구현하였다. 하지만, LKM의 특성상 랩퍼 모듈이 적재되기 전에 존재하는 프로세스들에 대한 audit id는 의존할 수 없는 정보가 저장되게 된다. 이에 대한 보완책으로 LxBSM을 적용한 이후 시스템에 대한 리부팅 과정을 통해 해당 정보를 유효화하는 것이 바람직하다[1].

LxBSM의 랩퍼 모듈의 is_selection_mask()를 통해 해당 시스템 콜에 대한 감사 자료 생성에 대한 마스킹이 가능하다. 감사 시스템이 켜져 있는지, 어떤 이벤트에 대해 감사 자료를 생성할 건지, 감사 자료가 생성되어야 하는 이벤트가 성공/실패/또는 두 경우 모두에 대해 감사 자료를 생성할 건지를 마스킹한다. 이 후 모든 이벤트에 공통되는 형식의 감사 자료-헤더 포맷-를 만들고 각 이벤트에 따라 다른 토큰(token)들-예, subject token, arg token 등-을 생성한다[14]. 각 이벤트에 대한 감사 자료 생성이 완료되면 드라이버 모듈의 감사 자료 처리 루틴인 audit_handler()를 호출하여 드라이버 모듈에 구현된 감사 자료 처리 루틴을 통해 사용자 모드로 전달하도록 구현되었다.

4.3. 데몬 모듈 구현

데몬 모듈은 응용 프로그램으로 구현되었다. auditon() 시스템 콜을 통해 감사 시스템을 On/Off 시킬 수 있다[2].

감사 데몬 프로세스는 감사 관련 설정 파일인 /etc/security/audit_*를 읽어 /dev/auditor에 대해 ioctl()를 통해 설정 정보-감사 시스템 On/Off 여부, 감사 이벤트 마스크 정보, 할당할 버퍼의 양, 데드락(dead lock)을 막기 위한 감사 데몬 프로세스 자체의 프로세스 ID정보, 감사 파일이 저장되는 파일 시스템의 최소 가용 용량에 대한 정보 등-를 드라이버 모듈과 랩퍼 모듈에 전달한다.

감사 데몬 프로세스는 mmap() 호출을 통해 파일 모드로 동작 시 드라이버 모듈의 버퍼에 생성된 감사 자료를 데몬 프로세스의 영역으로 이동시킨다.

5. 실험

LxBSM에 대한 실험은 성능과 안정성 두 가지 측면에서 수행하였다. 성능 실험을 통하여 LxBSM이 동작하는 시스템이 그렇지 않은 시스템과 비교해 어느 정도의 오버헤드를 유발하는지 측정하였다. 또한, LxBSM이 동작하는 시스템의 경우 시그널 방식과 피포 방식간의 성능 비교를 실험하였다. 성능 실험은 LMBench를 이용한 마이크로 벤치마크(Microbenchmark)와 Webbench를 이용한 매크로 벤치마크(Macrobenchmark)를 수행하였다.

안정성 실험을 통하여 커널 수준에서 동작하는 LxBSM이 시스템에 적용되어 안정성 측면에서

---

1) 물론 이 정보의 정확성을 엄밀히 요구하는 사이트의 경우

2) auditon() 시스템 콜은 LxBSM에 의해 리눅스에 새로 추가된 시스템 콜이다

문제를 일으키지 않는다는 것을 확인하였다. 이를 위해선 LTP(Linux Test Project)를 사용하였다.

실험은 레드햇 리눅스(Redhat Linux) 6.2[1] 버전을 인텔 펜티엄 III 933 MHz 시스템, 256MB 램 메모리, 256K 캐시를 가진 시스템에 설치한 후 수행하였으며, 캐시 효과등을 감안하여 10번의 실험 결과를 평균치로 환산하였다.

실험은 LxBSM만 적용하지 않고 상용 호스트 기반 침입탐지 시스템[4]을 연동하는 상황에서 측정함으로써, LxBSM이 적용되어 침입 탐지를 수행하는 실제 상황에서 발생하는 오버헤드를 측정결과로 제시한다. 왜냐하면, 감사자료를 생성하고 처리하지 않는 경우의 오버헤드는 미미하며 실제 상황에선 의미를 가지고 있자 않다고 생각하기 때문이다. 따라서, 제시된 모든 결과는 LxBSM에서 감사 자료를 생성하고 이를 침입 탐지 에이전트에서 처리하는 상황에서 측정되었다.

LxBSM의 초기 버전에선 생성된 감사 자료의 전달을 시그널 방식에 의존하는 시스템 -SNARE, Linux BSM-과 동일한 방식을 사용하였다. 하지만, 시그널 방식에 의존할 경우 감사 자료의 전달과정에서 빈번히 발생하는 사용자 모드와 커널 모드사이의 스위칭 및 감사 데몬과 감사 자료 생성 프로세스간의 컨택스트 스위칭으로 인해 오버헤드가 심각하였다. LxBSM의 이후 버전에서 피포 방식으로 접근하면서 이 오버헤드를 상당히 줄일 수 있었으며 결과를 통해 향상된 점을 보이겠다.

5.1. 성능 실험1: 마이크로 벤치마크

LMBench 2.0을 사용하여 첫 번째 성능 실험을 수행하였다. 결과중 일부의 경우 LxBSM을 사용한 경우 더 나은 성능 결과를 보이고 있다. 이는 캐시 효과상의 오차로 판단된다. 전체적인 결과를 통해 LxBSM을 통해 감사 자료를 생성하고 침입 탐지를 수행하더라도 시스템에 대해 그리 큰 오버헤드를 미치지 않음을 알 수 있다[표 1].

open/close의 경우 약 2배에 해당하는 오버헤드가 발생하는 것을 알 수 있으며, 이는 패스 토큰(Path Token)과 어트리뷰트 토큰(Attribute Token)[14] 생성시 가변 버퍼를 할당하는 과정에서 유발하는 오버헤드이다. 각 토큰 생성시 각각 약 3 마이크로 세컨드씩의 오버헤드를 유발하는 것을 확인하였다. 이는 각 토큰 생성시 현재 가변 버퍼를 동적으로 할당하는 과정에서 유발된다. 향후 가변 버퍼를 동적으로 할당하는 대신 제한된 길이의 고정 버퍼를 사용토록 함으로써 이 부분에 대한 더 나은 성능 향상을 기대할 수 있다.

---

1) 리눅스 커널버전 2.2.14

[표 1] LMBench 실험 결과: 시간 단위(microsecond)

테스트 항목	리눅스 커널 2.2.14	2.2.14+ lxbsm (SIGNAL)	2.2.14+ lxbsm (FIFO)	% Overhead (FIFO)
Simple syscall	0.3083	0.3276	0.3276	6.26
Simple read	0.4779	0.5213	0.4985	4.31
Simple write	0.3987	0.4280	0.4211	5.61
Simple stat	2.8002	2.8232	2.7951	-0.18
Simple fstat	0.6207	0.6374	0.6262	0.89
Simple open/close	3.5641	73.0492	10.7748	202
Select on 10 fd's	2.0362	2.0779	1.9531	-4.08
Select on 100 fd's	5.9518	6.976	5.9718	0.34
Select on 250 fd's	12.918	13.9636	12.9366	0.14
Select on 500 fd's	23.7414	27.3276	24.2423	2.11
Select on 10 tcp fd's	2.9734	3.1649	2.9252	-1.62
Select on 100 tcp fd's	17.7115	19.0837	17.7428	0.18
Select on 250 tcp fd's	47.75	51.3383	49.7117	4.11
Signal handler installation	0.936	0.95	0.95	1.50
Signal handler overhead	1.763	1.757	1.757	-0.34
Pipe latency	3.526	3.8068	3.6554	3.67
Process fork+ exit	272.1053	412.4211	272.4211	0.11
Process fork+ execve	1170	7442.75	1193	1.97
Process fork+ /bin/sh -c	4690.5	18445.5	4966	5.87
Pagefaults on /usr/tmp/XXX	473	492	488	3.16

5.2. 성능 실험 2: 매크로 벤치마크

매크로 벤치마크를 위해 WebBench 4.0.1을 사용하였다. LMBench와 동일하게 침입 탐지 시스템과 연동한 실제 상황에서의 결과를 시그널 방식과 비교해 보이겠다.

WebBench의 경우 아파치(Apache) 웹 서버[1]를 사용하였으며 각 클라이언트에서 보낸 HTTP request의 초당 처리 숫자와 throughput(bytes/sec)을 측정 결과로 제시한다[표2]. 시그널 방식으로 처리한 것과 상당한 차이가 있음을 알 수 있다. LxBSM이 적용되어 침입 탐지 시스템과 연동이 되는 경우에도 두 가지 테스트 항목에 대해 최소 82프로 이상의 성능을 보이는 것을 알 수 있다.

1) 버전 1.3.12-2

[표 2] WebBench 결과: throughput(bytes/second)

클라이언트 수	리눅스 커널 2.2.14		2.2.14+ lxbsm (SIGNAL)		2.2.14+ lxbsm (FIFO)	
	req/sec	throughput	req/sec	throughput	req/sec	throughput
1	243.53	1477171	109.74(45%)	688142(47%)	215.70(89%)	1304031(88%)
4	245.68	1461243	109.99(45%)	650141(44%)	221.57(90%)	1359173(93%)
8	241.77	1485054	103.78(43%)	656730(44%)	220.25(91%)	1338374(90%)
12	237.77	1436790	101.32(43%)	590302(41%)	218.01(92%)	1301478(91%)
16	241.80	1487218	102.39(42%)	591137(40%)	220.75(91%)	1326562(89%)
20	242.80	1476472	97.51(40%)	611842(41%)	206.16(85%)	1213080(82%)
24	243.80	1478349	98.99(41%)	609238(41%)	204.89(84%)	1224552(83%)
28	243.18	1461685	96.00(39%)	576100(39%)	218.80(90%)	1320484(90%)
32	242.51	1452152	93.68(38%)	594031(41%)	216.51(89%)	1309166(90%)
36	242.05	1424995	89.08(36%)	484729(34%)	222.45(92%)	1344214(94%)
40	240.98	1451410	92.24(38%)	508687(35%)	231.69(96%)	1393624(96%)
44	243.75	1478223	90.45(37%)	543521(37%)	229.39(94%)	1362228(92%)
48	246.61	1496245	89.88(36%)	514784(34%)	228.91(93%)	1383504(92%)
52	247.44	1491516	87.54(35%)	530134(36%)	230.37(93%)	1387629(93%)
56	240.40	1445033	85.88(36%)	545509(38%)	230.88(96%)	1401990(97%)
60	239.75	1473071	83.68(35%)	480541(33%)	228.80(95%)	1380656(94%)

5.3. 안정성 실험

시스템 콜(system call)을 후킹(hooking)하는 기법을 사용하는 경우 커널의 오리지널 시스템 콜 루틴에서 수행하는 에러 상황 체크 및 퍼미션 체크를 중복하여 수행해야 할 경우가 있다. LTP 툴을 사용하여 각 시스템 콜에 대한 테스트 중 시스템 콜에 대해 비정상적인 인자를 넘겨주고 이에 대해 적절한 에러 코드와 리턴값을 리턴하는 지에 대한 확인 할 수 있다. 예를 들어, execve()의 인자 중 파일네임(filename)에 -1이나 프로세스 영역 밖의 값이 넘어갈 경우, EFAULT를 에러 값으로 리턴해야 되는데 랩퍼 루틴[1)]에서 이를 체크하지 않으면 시스템에서 Oops메세지가 발생한다[2].

비정상적인 인자를 넘겨받은 LxBSM의 여러 랩퍼 루틴들에서 이에 대한 처리 루틴을 추가하여 정상적인 리턴 값과 에러 코드 세팅을 수행하도록 수정하였다. SNARE나 LinuxBSM과 같은 경우 LTP 테스트에 의해 패스하지 못하는 경우가 발생하며 이에 대한 보완이 필요하다.

6. 향후 연구 계획

LxBSM의 드라이버 모듈은 감사 자료를 처리하는 데 있어서-피포 모드의 경우- 리눅스 커널의 비선점성을 전제하여 하나의 감사 이벤트가 발생할 때마다 이를 피포 리더에 전달하고 버퍼를 리셋하도록 구현되었다. 리눅스 커널이 선점형 커널로 바뀌었을 때 이는 하나의 감사 레코드에

1) lxbsm_exeve()

다른 이벤트가 덮어쓸 수 있는 가능성이 존재할 수 있으므로 이에 대한 보완책이 필요하다.

LxBSM에선 랩퍼 모듈과 드라이버 모듈이 각각 내부적으로 버퍼를 할당하여 감사 자료를 생성하여 저장하고 있다. 이 버퍼를 공유하도록 함으로써 성능을 향상시킬 수 있다고 본다. 또한, open/close의 경우에서 보듯 동적으로 할당하는 방식의 오버헤드를 정적으로 할당함으로써 성능 향상이 가능하다.

LSM(Linux Security Module)[1]은 리눅스 운영체제에 대한 다양한 보안 정책을 융통성있게 지원하기 위해 커널 패치 형식으로 제안된 범용 프레임워크이다. LSM 프레임워크와 연동하는 LKM 기반의 보안 모듈은 접근 제어 용도가 주를 이루고 있다- SELinux[10], DTE Linux[11], Openwall[13], LIDS[8]-. LKM 기반의 감사 용도의 보안 모듈도 LSM 프레임워크와 연동될 수 있는지, 연동된다면 현재의 LSM에서 제공하는 여러 hook들과 security field들을 이용할 수 있는지 등에 대한 연구를 계획하고 있다.

7. 결론

LxBSM은 리눅스 운영체제에 대해 기존 커널의 변경하지 않고 LKM 기법을 통해 동적으로 로딩이 가능하도록 설계되었기 때문에 다운 타임 없이 타겟 시스템에 적용될 수 있다. 그리고, 성능과 안정성 면에서 LxBSM이 시스템에 미치는 영향을 최소화하기 위해 커스텀 피포 드라이버와 에러 확인 코드들을 구현하였다. 또한, 침입 탐지 시스템과 같은 외부의 프로세스가 용이하게 생성한 감사 자료를 활용할 수 있는 인터페이스를 가지고 있으며, 실제 연동을 통해 CERT나 SecurityFocus등에서 보고된 리눅스 운영체제에 대한 침입 시도의 탐지의 가능성을 확인하였다. LxBSM은 SunSHILED BSM에서 생성하는 이벤트의 대부분을 생성함으로써 높은 수준의 보안성이 요구되는 경우에 활용될 수 있는 장점을 지닌다.

인터넷을 통한 손쉬운 정보 공유를 통한 정보 보호 침해 사고등은 날로 증가일로에 있다. 리눅스 운영체제는 전 세계의 개발자들이 상호 협력하여 개발하고 있는 공개 소스 기반의 운영체제다. 따라서, 유사 운영체제에 비해 더욱 안전한 보안 기능이 구비되어야함에도 불구하고 감사 자료 생성 기능에 있어서만큼은 매우 취약한 상황이다. 즉, 리눅스 운영체제에 대한 C2 레벨 이상의 보안성을 만족시키기 위한 기본 기능 중의 하나인 감사 자료 생성 기능은 접근 제어나 기타 공개 프로젝트 등에서 성공적인 결과를 보여주고 있지 못하다.

본 논문의 결과를 통해 리눅스 운영체제상에서도 유사 운영체제와 동일한 수준의 감사 자료 생성기능이 LxBSM 개요부분에서 언급한 설계 목적을 만족하면서 추가될 수 있는 가능성을 확인하였다.

참고 문헌

[1] C. Wright and C. Cowan, S. Smalley, J. Morris, G. Kroah-Hartman, "Linux Security Modules: General Security Support for the Linux Kernel," In *Proceedings of 11th USENIX Security Symposium*, August 2002.

[2] D. Bovet and M. Cesati, Understanding the Linux Kernel, O'Reilly, 2001.

[3] D. Ghormley, D. Petrou, S. Rodrigues, and T. Anderson, "SLIC: An extensibility system for commodity operating systems," In *Proceedings of USENIX Annual Technical Conference*, June 1998.

[4] Inzen Corporation, http://www.inzen.com.

[5] L. McVoy and C. Staelin, "lmbench: Portable Tools for Performance Analysis," In *Proceedings of USENIX Annual Technical Conference*, January 1996.

[6] Linux BSM, http://sourceforge.net/projects/linuxbsm/.

[7] Linux Documentation Project, http://www.linuxdoc.org.

[8] Linux Intrusion Detection System, http://www.lids.org.

[9] M. Barabanov and V. Yodaiken, "A Real-Time Linux," In *Proceedings of the Linux Applications Development and Deployment Conference*, January 1997.

[10] P. Loscocco and S. Smalley, "Integrating Flexible Support for Security Policies into the Linux Operating System," In *Proceedings of USENIX Annual Technical Conference*, April 2001.

[11] S. Hallyn and P. Kearns, "Domain and Type Enforcement for Linux," In *Proceedings of the 4th Annual Linux Showcase and Conference*, October 2000.

[12] SNARE, http://intersectalliance.com/projects/Snare/.

[13] Solar Designer, "Non-Executable User Stack," http://www.openwall.com/linux/.

[14] "SunSHIELD Basic Security Module Guide," SUNSOFT, 1995.

[15] The Linux Test Project, http://ltp.sourceforge.net.

[16] "Trusted Computer Security Evaluation Criteria," DOD 5200.28-STD, Department of Defense, 1985.

[17] Ziff-Davis WebBench, http://zdnet.com.