제 18 회 한국정보처리학회 추계학술발표대회 논문집 제 9 권 제 2 호 (2002.11)

보안 운영체제의 성능 평가에 대한 연구

홍철호*, 고영웅*, 김영필*, 신용녀**, 김정녀***, 유혁*
*고려대학교 컴퓨터학과
**한국정보보호진흥원
***한국전자통신연구원

e-mail : *chhong@os.korea.ac.kr, *yuko@os.korea.ac.kr, *ypkim@os.korea.ac.kr,
**ynshin@kisa.or.kr, ***jnkim@etri.re.kr, *hxy@os.korea.ac.kr

A Study on Performance Evaluation of Secure Operating Systems

Cheol-Ho Hong*, Young-Woong Ko*, Young-Pill Kim*, Yong-Nyu Shin**,
Jeong-Nyu Kim***, Chuck Yoo*
*Deptartment of Computer Science and Engineering, Korea University
**Korea Information Security Agency
***Electronics and Telecommunications Research Institute

요 약

일반적인 운영체제에 보안 기능을 강화하기 위하여 접근 통제, 침입 탐지 그리고 감사 및 추적 등과 같은 새로운 보안 기능이 계속해서 추가되고 있다. 추가된 보안 기능에 의해서 어느 정도 시스템 성능에 변화가 생기는지는 보안 운영체제를 도입하려는 사용자들에게 중요한 선택 기준이 되고 있다. 그러나 현재 개발되고 있는 다양한 보안 운영체제들이 어느 정도의 성능을 보여주고 있는지에 대한 분석 자료는 거의 없다. 본 논문에서는 잘 알려진 보안 운영체제를 대상으로 성능 측정 결과를 보이고 있으며, 오버헤드가 발생하는 주요 원인에 대해서 분석을 하고 있다. 분석에 사용된 보안 운영체제는 공개 소스에 기반한 SELinux 이며, 보안 운영체제를 분석하는데 사용한 벤치마크 툴은 유닉스 성능 분석에 널리 활용되고 있는 lmbench 이다. 본 논문에서 제시하는 실험결과는 보안 운영체제를 도입하려는 사용자들과 보안 운영체제를 개발하는 개발자들에게 유용한 정보로 사용될 수 있다.

1. 서론

전통적인 운영체제는 네트워크로 연결된 개방 시스템에 대한 심각한 고려 없이 설계되어 있으며 운영체제 및 응용 프로그램의 허점을 통해서 해커 또는 바이러스로부터 공격 받을 때마다 필요한 대책을 세우는 수동적인 개념이었다. 따라서 외부로부터의 위협요소 및 내부적 정보 남용과 같은 문제점에 대해서 적절한 해결책을 제공해줄 수 없으며 시스템을 통제하고 관리해주는 운영체제 수준에서 자원들에 대한 강제적 접근 통제 및 사용자에 대한 감사 및 추적 기능 등을 필요로 하고 있다.

보안 운영체제란 운영체제 상에 내재된 각종 보안 결함을 제거하고 안전하게 시스템을 보호할 수 있도록 접근통제, 사용자 인증, 감사추적 및 침입 탐지 등의 보안기능이 추가된 안전한 운영체제를 의미한다[1]. 이러한 보안 운영체제는 어떠한 경우에도 그 기능이 정지되어서는 안 되는 국가 기간 망과 군수시설을 통제하는 시스템에서 필수적이다. 또한 기업 간의 거래, 금융 서비스 등 정보 시스템의 안전함을 우선해야 하는 상황에서도 보안 운영체제는 그 필요성이 절실히 요구되고 있다.

보안에 대한 중요성이 점점 높아지면서 전 세계에 있는 수많은 연구기관 및 회사에서 보안 시스템에 대한 연구를 진행하고 있으며, 일부 연구 결과물은 제품

이 연구는 한국정보보호진흥원과 전자통신연구원의 지원으로 진행되었습니다.

으로 판매되고 있다. 하지만 보안 운영체제가 보안에 대한 문제를 해결해줄 수 있는 주요한 해결책임에도 불구하고 실제적으로 널리 사용되고 있지 못하다. 주된 이유를 몇 가지 언급하자면 다음과 같다. 첫째, 아직까지 보안 운영체제의 중요성에 대한 인식이 널리 확산되지 못하였다는 것을 언급할 수 있다. 즉 대부분의 시스템 관리자와 사용자들은 보안 운영체제를 도입했을 때 어떤 효과를 얻을 수 있을 것인지에 대해서 정확한 정보를 가지고 있지 못하며, 보안에 대한 인식이 부족해서 적극적으로 보안 침해에 대응하려는 태도를 가지고 있지 않다. 두 번째는 보안 운영체제를 사용했을 때 기존의 응용 프로그램과의 호환성이 떨어진다는 것을 언급할 수 있다. 기존에 사용하고 있던 수많은 응용 프로그램 중에서 일부 또는 많은 프로그램들이 보안 운영체제상에서 그대로 수행되지 못하고 변경되어야 하는 문제점이 있다. 세 번째는 보안 운영체제를 도입했을 때, 시스템 관리를 하기 위한 교육이 필요하며, 새로운 환경에 대해서 부담감을 가지는 관리자 및 사용자들은 기존의 시스템을 그대로 사용하려는 성향을 보이게 된다. 마지막으로 중요한 이유 중의 하나는 보안 운영체제에 대한 성능이 제대로 입증되지 않았다는 사실이다. 따라서 보안 운영체제가 보편적으로 사용될 수 있게 하기 위해서는 보안 운영체제의 성능에 대한 분석이 필수적이다.

본 논문은 리눅스 쪽의 대표적인 접근통제 프로젝트의 하나인 SELinux(Security-Enhanced Linux)를 대상으로 보안 운영체제의 성능 평가를 진행하였다. SELinux 는 일정 수준의 보안 기능을 제공하고 있으며 소스가 공개되어 있어 성능 평가 결과를 분석하는데 용이하다. 한편 성능 분석을 위한 평가 도구는 유닉스 벤치마크에 널리 사용되는 lmbench 이다.

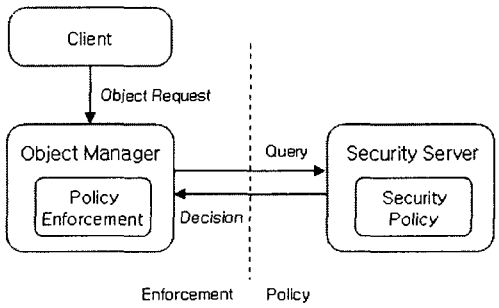
본 논문의 구성은 다음과 같다. 2 장에서는 관련 연구로 성능 분석 대상인 SELinux 와 성능 평가 도구인 lmbench 에 대해 기술한다. 3 장에서는 lmbench 를 이용하여 SELinux 의 성능 평가를 하고 평가 결과를 분석한다. 마지막으로 4 장에서 결론을 맺는다.

2. 관련 연구

2.1 SELinux

SELinux 는 NSA(National Security Agency)의 주도하에 여러 연구소에 의해 구현된 보안 리눅스 운영체제이다. SELinux 에는 접근통제 중 강제적 접근통제인 MAC(Mandatory Access Control)이 구현되어 있는데 전통적인 강제적 접근통제의 구현사항보다 더 진보된 것이다. SELinux 는 전통적인 MAC 의 문제점을 극복하기 위해 좀더 유연한 방법을 제공하였는데 그것은 보안 정책 집단과 그것을 행하는 메커니즘을 분리시켜서 다양한 MAC 정책을 가능하게 한 것이다. SELinux 는 프로세스와 파일, 그리고 소켓에 대해 MAC 의 기능을 제공하고 있고, 또한 NAI Labs 의 도움을 받아 procfs 과 devpts 파일 시스템 등 추가적인 커널 기능에 대해서도 MAC 정책을 구현하고 있다.

SELinux 는 Flask 구조를 가져왔으므로 SELinux 의 보안 구조는 Flask 구조와 흡사하다[2]. Flask 구조의 특징은 어떠한 운영체제에도 적용 가능하게끔 보안 구조만을 정의하고 있다는 것이다. 그리고 유연한 보안 정책의 지원을 위해서 보안 정책과 보안 정책을 수행하는 보안 메커니즘과의 엄격한 구분을 두어서 최대한의 유연성을 보장해 주고자 한다. 아래 [그림 1]은 관련 구조를 나타내고 있다.



[그림 1] SELinux 의 구조

[그림 1]에서 주체인 클라이언트로부터 객체에 대한 접근 요청이 들어오면 객체 관리자가 보안 서버에 객체에 대한 접근 권한 여부를 물어보게 되고 보안 서버는 그 객체에 대한 접근 결정을 객체 관리자에게 넘겨주는 Flask 의 보안 구조를 보이고 있다. 객체 관리자는 접근 결정에 의해 보안 정책을 수행하게 된다. Flask 구조는 마이크로 커널에 기반을 두고 있으므로 보안 메커니즘을 수행하는 컴포넌트를 보안 서버라고 부르고 있다.

2.2 lmbench

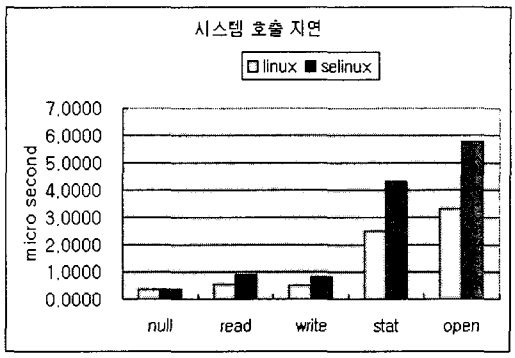
lmbench[3]는 일반적으로 유닉스 시스템의 성능 분석을 위해서 주로 사용되는 벤치마크이며, 유닉스 시스템 상의 병목지점을 측정해준다. 시스템 지연(latency)이나 프로세서와 메모리 네트워크 간의 데이터 대역폭(bandwidth)을 측정할 수 있는 여러 개의 작은 프로그램들로 구성되어 있다. lmbench 는 지연과 대역폭을 측정하는데 중점을 두고 있는데 대부분의 성능 문제가 지연 문제나 대역폭 문제 또는 그 두가지 요소의 혼합적인 문제로 생기는 경우가 많기 때문이다.

3. SELinux 의 성능 평가

성능 분석에 사용된 일반적인 리눅스 커널은 2.4.7 버전이며, SELinux 는 2.4.17 버젼을 사용하였다. 성능 분석에 사용된 하드웨어 시스템 사양은 Pentium III 866MHz, 256M RAM 의 데스크탑 PC 이다. 실험 카테고리는 보안 기능의 추가로 발생하는 지연과 대역폭에 대한 실험이며, 세부적으로 시스템 호출 지연, 파일 시스템 지연, 프로세스 지연, 메모리 대역폭 등이다. 이때, SELinux 의 보안 정책은 기본적으로 제공하

는 예제를 이용하였다.

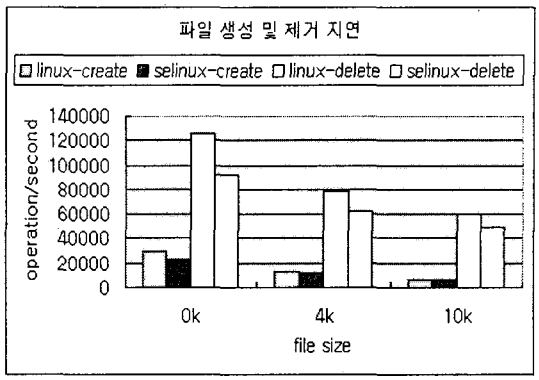
3.1 시스템 호출 지연



[그림 2] 시스템 호출 지연 비교

첫 번째 벤치마크는 시스템 호출을 수행하는데 있어서 일반 리눅스와 SELinux 간의 성능을 비교하는 것이다. lmbench 에서 제공해주는 시스템 호출 지연 분석은 다섯 가지 옵션이 있으며, 각각 null, read, write, stat 그리고 open 이다. 이중에서 null 옵션은 /dev/null 에 한 바이트를 써넣는데 걸리는 시간을 측정하며, 나머지 옵션은 파일 시스템에 대한 시스템 호출이다. 실험 결과에서 알 수 있듯이 SELinux 는 파일에 접근하는 오퍼레이션이 많은 시스템 호출을 수행할 때 가장 큰 성능상의 하락을 보여주고 있다. null 시스템 호출의 경우 /dev/null 디바이스를 접근할 때 보안 정책을 점검하게 되며, 이때 일반 리눅스보다 약간의 지연을 발생시키게 된다. read 시스템 호출의 경우는 리눅스 커널 내부에서 read 오퍼레이션이 수행되는 과정이 여러 번의 접근 제어 확인 과정을 필요하게 되므로 null 시스템 호출에 비해서 대략 0.3 마이크로 세컨드 정도의 시간이 더 소요되고 있다. 구체적으로 read 에서 소요되는 과정은 파일을 한 블록 읽어내기 위해서 파일 시스템에 있는 슈퍼블럭에서 i-node 를 읽어오고, i-node 에서 해당 블록의 위치를 찾아내는 과정 그리고 실제적으로 저 수준의 디스크 디바이스 드라이버에게 블록 요청을 보내는 여러 단계의 과정으로 이루어져 있다. 각 과정에서 발생하는 접근 제어 정책은 null 시스템 호출에 비해서 상당한 지연을 초래하게 된다. write 시스템 호출은 read 시스템 호출과 비슷한 과정을 거치게 되므로 수행 결과가 비슷하게 나오고 있음을 알 수 있다. stat 는 read 또는 write 에 비해서 더 많은 지연이 발생함을 알 수 있다. 리눅스 커널 내부에서 stat 이 하는 작업은 파일의 상태를 읽어서 메모리의 특정한 부분에 정보를 기록하는 작업을 하는 것이다. 이때 파일의 상태에는 각 소유자에 대한 정보 및 파일의 논리적인 정보들이 기록되어 있으며, SELinux 는 파일 정보를 기록할 때, 부가적인 보안 항목들을 관리하기 때문에 stat 의 지연이 높게 나오는 것이다.

3.2 파일 시스템 지연



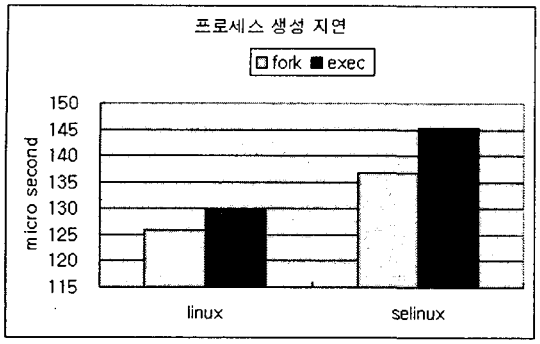
[그림 3] 파일 생성 및 제거 지연 비교

두 번째 실험 결과는 매초 해당되는 파일 사이즈 만큼의 파일 개수를 생성하고 지우는데 걸리는 시간을 측정한 것이다. 이때 X 좌표는 파일의 크기를 변화시키면서 측정했음을 보이고 있으며, Y 좌표는 단위 시간에 수행한 오퍼레이션의 개수를 의미한다. linux-create 는 일반 리눅스에서 일초 동안 생성한 파일의 개수를 보이고 있으며, selinux-create 는 SELinux 에서 일초 동안 생성한 파일의 개수를 나타낸다. 그리고 linux-delete 는 일반 리눅스에서 일초 동안 제거한 파일의 개수를 보이고 있으며, selinux-delete 는 SELinux 에서 일초 동안 제거한 파일의 개수를 나타낸다.

본 실험 결과가 의미하는 바는 SELinux 가 파일을 생성 및 삭제할 때 파일시스템에 추가적으로 기록하는 보안 레이블과 같은 보안 요소에 의해서 추가되는 시간이 어느 정도인지를 보여주고 있다. 파일 생성 실험에서 파일의 크기가 0Kbyte 일 때, 일반 리눅스에 비해서 SELinux 는 대략 80 퍼센트정도의 성능을 보이고 있으며, 10Kbyte 일 때, 94 퍼센트의 성능을 보인다. 파일 제거 실험에서 파일의 크기가 0Kbyte 일 때, 일반 리눅스에 비해서 SELinux 는 대략 74 퍼센트정도의 성능을 보이고 있으며, 10Kbyte 일 때, 83 퍼센트의 성능을 보인다. 따라서 파일의 크기가 커질수록 SELinux 의 성능은 일반 리눅스의 성능과 별 차이가 없어지고 있음을 알 수 있다. 이러한 결과가 나오는 이유는 파일의 크기가 커질수록 보안 정책을 처리하는데 소요되는 지연보다 실제적으로 파일을 생성 및 제거하는데 소요되는 지연이 더 커지기 때문이다.

따라서 작은 파일을 생성 및 제거하는 작업이 빈번한 시스템에서는 보안 운영체제의 오버헤드가 상당히 크게 작용하게 되며, 파일의 생성 및 삭제가 빈번하지 않고, 대용량의 데이터를 생성 및 삭제하는 경우, 보안 운영체제에 의해서 지연되는 시간은 미미하게 된다.

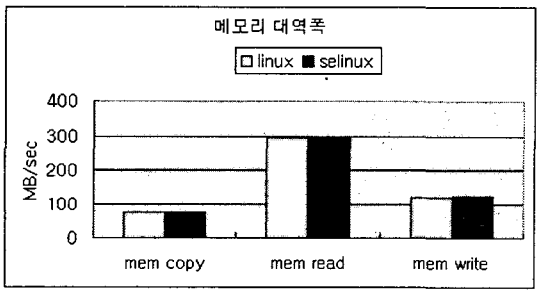
3.3 프로세스 지연



[그림 4] 프로세스 생성 지연 비교

세 번째 실험에서는 프로세스의 생성에 소요되는 지연을 상호 비교 하고 있다. lmbench 에서는 프로세스를 생성할 때 몇 가지 옵션을 선택할 수 있다. 첫 번째는 fork 옵션이며, 현재 수행되는 프로세스와 동일한 이미지를 가지는 또 다른 프로세스를 생성시키고 종료시키는데 걸리는 시간을 측정한다. 두 번째는 execve 옵션이며, 이것은 현재 수행되는 프로세스와 동일한 프로세스를 생성시키고 곧바로 다른 프로세스 이미지를 수행시키는 방식이다. 본 실험에서 SELinux 는 일반 리눅스보다 대략 10 퍼센트 정도 추가 지연이 발생하였다. 이러한 결과가 나오는 이유는 fork 및 execve 를 사용한 프로세스 생성 과정 중간에 파일 시스템에서 프로세스의 이미지를 찾아내고 보안 정책을 통하여 접근 허가를 얻어내는 과정이 추가되었기 때문이다.

3.4 메모리 대역폭



[그림 5] 메모리 대역폭 비교

본 실험에서는 메모리 대역폭에 대한 성능 분석을 보이고 있다. lmbench 에서 메모리 대역폭을 측정하는 bw_mem 는 세가지의 옵션을 제공하며, 이중에서 cp 옵션은 지정한 양만큼의 메모리를 0 으로 초기화하고 메모리의 반을 나머지 반에 복사하는 시간을 측정한다. rd 옵션은 지정한 양만큼의 메모리를 0 으로 초기화하고 정수 단위로 읽고 더하는 시간을 측정한 값이다. 마지막으로 wr 옵션은 지정한 양만큼의 메모리를 0 으로 초기화하고 증가하는 4 바이트 정수를 메모리에 저장하는 시간을 측정한 값이다. 이때, X 축은 옵션을 의미하고 Y 축은 초당 이동된 메가 바이트를 나타낸다. 실험 결과에서 메모리에 대한 읽기, 쓰기, 복사와 같은 오퍼레이션에 대해서 SELinux 와 일반 리눅스는 거의 차이가 없음을 알 수 있다. 그 이유는 메모리 객체에 대한 접근 제어 정책이 현재 지원되지 않기 때문이다.

4. 결론

본 논문에서는 lmbench 를 이용하여 보안 운영체제인 SELinux 의 성능 평가를 수행하였으며 이를 통하여 SELinux 가 기존의 리눅스에 비해 어느 정도의 성능 차이를 보이는지 분석해 보았다. 실험결과 시스템 호출의 경우 null, read, write 의 경우 일반 리눅스와 SELinux 간에 차이가 크게 나지 않았지만 보안 항목들을 자주 접근해야 하는 stat 과 open 의 경우 70% 정도의 오버헤드가 발생했다. 파일 시스템 지연의 경우 작은 파일을 자주 생성하고 삭제하는 경우 SELinux 의 성능이 저하되는 것으로 나타났다. 프로세스 생성 지연의 경우 SELinux 가 일반 리눅스 보다 10%정도의 추가 지연이 발생하였으며, 성능의 변화가 생기는 부분은 프로세스의 이미지를 찾아내서 접근 허가를 얻어내는 과정이 추가되었기 때문이다.

본 논문의 결과로 보안 운영체제를 도입하려는 사용자들은 기존의 시스템에 비해서 어느 정도 성능의 변화가 있는지에 대한 성능 비교 분석 결과를 유용한 정보로 사용할 수 있을 것이다. 본 논문에서 보이고 있는 분석 결과는 SELinux 에 한정되어 있으나, 향후 분석 대상 보안 운영체제를 확대하고 객관적인 성능을 분석할 수 있는 시나리오를 개발하는 것을 목표로 하고 있다.

참고문헌

[1] 한국정보보호진흥원, *안전한 OS 개발 선행 연구*, 1998. 12

[2] Peter Loscocco, "Integrating Flexible Support for Security Policies into the Linux Operating System", 2001 USENIX Annual Technical Conference, 2001

[3] Larry McVoy, "lmbench: Portable tools for performance analysis", USENIX, 1996

[4] D. P. Bovet, *Understanding the linux kernel*, O'Reilly, 2001