제20회 한국정보처리학회 추계학술발표대회 논문집 제10권 제2호 (2003. 11)

신뢰성 있는 커널 컴포넌트를 이용한 마이크로커널 기반 시스템 콜 인터포지션 기법

김영필*, 유혁*
*고려대학교 컴퓨터학과

e-mail : ypkim@os.korea.ac.kr

Microkernel-based System Call Interposition Mechanism using Reliable Kernel Component

Young-Pill Kim*, Hyuck Yoo*
*Department of Computer Science & Engineering, Korea University

요 약

시스템 콜 인터포지션 메커니즘은 침입 탐지 및 접근 제어와 같은 시스템 보안 기능을 강화하기 위해서 사용하는 방법이다. 현재까지 알려진 시스템 콜 인터포지션의 구현 방법은 크게 라이브러리 기반, 커널 기반 그리고 유저레벨 프로세스 기반의 접근방식이 있다. 기존의 방식들은 대부분 범용적인 모노리틱커널에 적용되어 사용되고 있으며, 최근에 보안 운영체제로 개발되고 있는 마이크로커널 방식의 시스템에서는 적합하지 않다. 본 논문에서는 기존에 연구되었던 시스템 콜 인터포지션 방법들을 소개하고 마이크로커널 기반의 시스템 콜 인터포지션을 위한 메커니즘과 기반이 되는 커널 컴포넌트들을 안전하게 관리하는 방법을 제안하고 있다. 제안된 메커니즘은 마이크로커널 위에 모니터 서버를 두고, IPC 가 수행될 때 특정 시스템 콜을 비동기 IPC 를 이용하여 감시하는 방식을 취하고 있다.

1. 서론

사용자가 응용프로그램 통해서 커널이 관리하는 시스템 자원들을 이용하기 위해서는 운영체제가 제공하는 시스템 콜을 호출하여야 한다. 시스템 콜은 정상적인 경우 응용프로그램 마다 특정의 패턴을 보이게 되는데, 이러한 정상적인 패턴과 비정상적인 경우의 패턴 분석은 침입 탐지를 위한 자료로 이용할 수 있다.[1] 이러한 분석이 가능하려면 시스템 콜을 실시간적으로 감시할 방법이 필요하다. 그리고 분석을 통해 침입 탐지가 되면 그 응용프로그램의 시스템 콜 요청은 거부해야 한다.

이와 같이 시스템 콜을 감시하고 분석 및 제한하는 메커니즘이 시스템 콜 인터포지션 기법이다. 이 방법을 통해서 시스템 콜의 파라미터 분석이나 감사 및 추적이 가능하며, 요청된 콜을 거부하거나 승인할 수 있다. 이러한 시스템 콜 인터포지션 방법에는 크게 세 가지의 접근 방법이 있으며, 각각, 라이브러리 기반, 커널 기반, 유저레벨 프로세스 기반으로 나눌 수 있다.

언급한 접근 방식들은 대부분 범용적으로 사용되는 UNIX 나 Linux 등과 같은 모노리틱커널 구조를 바탕으로 설계되었다. 그러나 모노리틱커널 구조가 아닌 마이크로커널 구조에서는 기존의 방식과는 다르게 시스템 콜 인터포지션 방법을 구현해야 한다. 마이크로커널 구조는 최근의 보안 운영체제가 채택하고 있는 방식으로 Synergy 프로젝트의 일환으로 진행된 DTOS[2]나 그 후속 프로젝트로 진행된 FLASK[3]등이 이에 해당한다.

본 논문의 내용은 다음과 같다. 2 장에서는 기존에 제시 되었던 시스템 콜 인터포지션 방법들을 기술하고, 3 장에서는 제안하는 마이크로커널기반 시스템 콜 인터포지션 기법의 특징 및 동작 방식에 대해서 설명한다. 4 장에서는 제시하는 방법과 기존 방법들과의 비

교 및 분석을 하고 마지막으로 5 장에서 결론을 맺는다.

2. 관련 연구

악의적인 사용자들의 공격 유형은 여러 가지가 있지만, 대다수의 방법들은 시스템이 제공하는 시스템 콜을 사용할 수밖에 없다. 예를 들어, UNIX 의 경우, 침입자는 루트 권한을 가지고 있는 쉘을 실행하거나, /etc 나 /var 등의 중요한 디렉토리 안의 파일을 수정하려 할 것이다. 프로그램을 실행하거나, 파일 조작을 하는 모든 과정은 시스템 콜을 거쳐야 한다. 따라서 외부 공격으로부터의 대응 및 탐지를 위해서 시스템 콜을 실시간적으로 감시해야 한다. 이를 위해서 시스템 콜이 발생했을 때, 시스템 콜의 파라미터를 검사하고 시스템 콜 요청을 수락하거나 거부할 수 있는 메커니즘이 필요하다. 이러한 메커니즘을 시스템 콜 인터포지션[4,5,6]이라고 한다. 그림 1 은 시스템 콜 인터포지션의 개념을 잘 보여주고 있다.

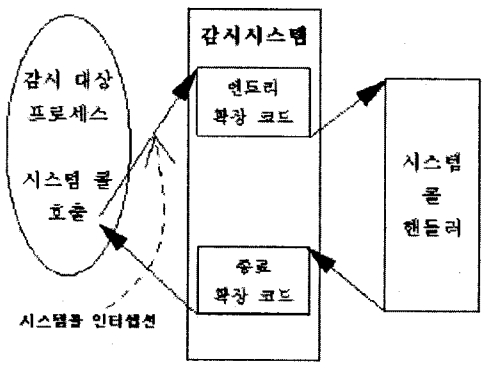

그림 1 System call interposition

유저 프로세스가 시스템 콜을 하게 되면 커널의 시스템 콜 핸들러가 실행되기에 앞서서 감시 시스템의 엔트리 확장 코드가 먼저 수행된다. 엔트리 확장 코드에서는 시스템 콜 요청을 수락할 것인지, 거부할 것인지 결정한다. 이때, 시스템 콜 핸들러는 커널 코드 안에 존재하므로 커널 영역의 주소공간을 사용하지만, 감시 시스템은 그 구현 방식에 따라서 다양한 위치에 놓일 수 있다. 시스템 콜 인터포지션 접근 방법을 분류하면, 라이브러리 기반의 접근방식[4], 커널 기반의 접근 방식[5] 그리고 유저레벨 프로세스 기반의 접근 방식[6]으로 나눌 수 있다.

1) 라이브러리 기반 접근 방식

라이브러리기반의 접근 방식은 감시 시스템의 확장 코드들이 시스템 콜의 래퍼(wrapper) 함수들을 수정하여 추가되는 방식이다. 일반적으로, 시스템 콜은 libc 와 같은 라이브러리 안의 래퍼 루틴에서 트랩과 같은 소프트웨어 인터럽트를 호출하게 된다. 따라서 시스템 콜 호출 루틴의 앞뒤에 코드를 쉽게 추가할 수 있다. 라이브러리 기반의 접근 방식이 가지는 장점은 커널을 수정하거나 커널 내부에 들어갈 필요가 없으므로 구현과 기능 확장성이 뛰어나다는 것에 있다.

2) 커널 기반 접근 방식

커널 기반 접근 방식은 시스템 콜 인터셉션을 운영체제 커널 안에 구현하고, 프로세스를 감시하기 위한 모든 확장 코드들이 커널 모드에서 수행되는 방식이다. 시스템 콜은 반드시 커널 내부에 있는 확장 코드를 거치게 되므로 시스템 콜을 중간에서 가로채는 오버헤드가 매우 적다.

3) 유저레벨 프로세스 기반 접근 방식

유저레벨 프로세스 기반 방식은 시스템 콜의 인터셉션을 커널에서 수행하고 감시시스템은 유저레벨 프로세스에서 구현하는 방법이다. 감시 대상이 되는 프로세스를 감시하기 위한 메커니즘은 대부분의 UNIX 시스템에서 제공하고 있는 ptrace, strace, /proc 등을 이용한다. 유저레벨 감시 시스템을 이용하여 침입 탐지에 대한 유연한 플랫폼을 제시하고 이를 통해서 확장성을 제공한다.

3. 마이크로커널 기반 시스템 콜 인터포지션 메커니즘

본 논문에서는 마이크로커널 상에서 수행하는 응용 프로그램의 시스템 콜을 감시하고 악의적인 공격을 찾아 낼 수 있는 시스템 콜 인터포지션 방법을 제안하고 있다.

본 연구에서 대상으로 하고 있는 시스템은 마이크로커널 구조의 운영체제이다. 마이크로커널은 물리 메모리나 하드웨어 등의 자원에 대한 저수준 관리, IPC(Inter Process Communication)와 동기화 등 운영체제의 필수기능만을 보유하고, 대부분의 운영체제 서비스들이 유저레벨의 서버로 분산된다. 이로 인하여 운영체제의 기능에 대한 확장성이나 유연성을 가진다. 마이크로커널 상에서 수행되는 서버들은 단일 서버 모델과 다중 서버 모델로 나눌 수 있다. 단일 서버 모델은 마이크로커널 위에서 UNIX 의 모든 기능을 갖춘 UNIX 서버가 동작하는 구조이다. 다중 서버 모델은 운영체제 서비스가 각각의 서버로 동작하는 구조이며, 스케쥴러 서버, 외부 페이저, 가상 메모리 관리 서버 등등이 존재할 수 있다.

본 논문에서는 단일 서버 모델 마이크로커널에서 시스템 콜 인터포지션 기법을 사용하는 것이다. 그림 2 는 본 논문에서 제시하는 마이크로커널 기반 시스템 콜 인터포지션 메커니즘을 보이고 있다

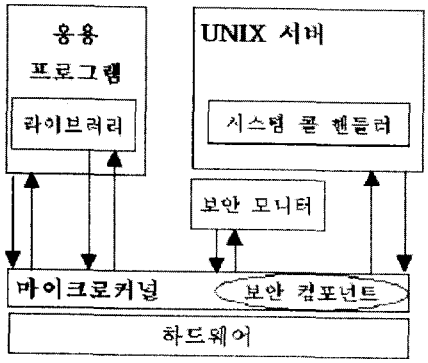

그림 2 마이크로커널 기반 구조

그림 2 에서 알 수 있듯이 응용 프로그램에서 발생하는 시스템 콜은 라이브러리를 통해서 IPC 로 변환되어 UNIX 서버로 전달하게 된다. 마이크로커널에게 UNIX 서버의 시스템 콜 서비스를 위한 IPC 요청이 들어오게 되면, 커널은 UNIX 서버에게 그 요청을 전달한다. 이와 더불어 기존의 시스템 콜 인터포지션 메커니즘에서 감시 시스템에 해당하는 보안 모니터에게 현재의 요청에 대한 정보를 함께 전달하게 된다. 이러한 전달은 보안 컴포넌트를 통해서 수행된다. 다음은 본 논문에서 제안하는 마이크로커널 기반 시스템 콜 인터포지션 메커니즘에서 사용하는 구성 요소들에 대해서 기술한다.

1. 보안 컴포넌트

마이크로커널에 내부에 있는 보안 컴포넌트는 응용프로그램으로부터 요청된 시스템 콜 요청을 보안 모니터와 UNIX 서버에게 전달하는 역할을 한다. 이러한 컴포넌트에게 요구되는 기능은 다음과 같다.

- 감시 대상 시스템 콜의 분류

패턴 분석을 하기 위하여 하나의 응용프로그램에서 호출하는 모든 시스템 콜에 대한 기록을 남기는 것은 성능의 저하를 유발할 수 있다. 따라서 감시 대상의 시스템 콜을 선정하고 해당 시스템 콜이 요청되었을 경우만 모니터 서버로 메시지를 보내야 한다.

- 중요한 서버의 보호

보안 기능의 많은 부분이 구현되는 보안 모니터나 UNIX 서버는 사용자 레벨의 서버이기 때문에 사용자가 임의적으로 죽일 수 없도록 제제를 할 수 있어야 한다. 따라서 kill 과 같은 시스템 콜을 통해서 중요한 서버를 죽이려고 한다면 이 요청을 거부해야 한다.

- 확장성과 신뢰성

보안 컴포넌트는 커널 컴포넌트이며 필요시 확장이 가능해야 하는데 특권레벨인 커널레벨에서 수행되므로 신뢰성 역시 보장되어야 한다. 확장성을 위해서 커널 컴포넌트를 사용자 레벨에서 구현하고, 실행 시에 커널 내에서 실행하는데 신뢰성을 주기 위해서 가상메모리에 기반한 고립화(isolation) 기법을 적용한다.[8] 이것은 커널 페이지 테이블의 일부를 사용하여 컴포넌트를 보호된 메모리 영역에서 실행하게 한것으로 컴포넌트 자체의 힙(heap)과 스택(stack), 그리고 몇 가지의 커널 객체들을 할당한 것으로 이루어져 있다. 컴포넌트와 커널간의 데이터 이동은 경량화한 컴포넌트 간 통신을 이용하며 컴포넌트가 커널 메모리에 직접 쓰기를 하는 것은 막아 오류나 버그에 의한 피해로부터 보호할 수 있게 된다.

2. 보안 모니터

보안 모니터가 해야 하는 기능은 시스템 콜의 감사 및 기록이다. 감사에 해당하는 사항은 시스템 콜의 파라미터 분석이 요구되고, 기록에 필요한 정보들은 시스템 콜이 발생한 시점과 요청한 응용프로그램의 정보, 그리고 서비스할 UNIX 서버의 정보 등이 된다. 이러한 기능은 파일 시스템과의 입출력을 요구하기 때문에, 자주 반복되는 것보다는 일정한 주기마다 한번씩 수행되는 것이 효과적이다. 따라서 보안 모니터 서버는 주기적으로 실행되는 사용자 레벨의 서버로 생성한다. 사용자 레벨의 서버로 실행되더라도 보안 컴포넌트에 의해 보호되므로 신뢰성을 보장할 수 있다.

3. 마이크로커널 중재의 시스템 콜 메커니즘

모노리틱커널 기반의 시스템에서는 시스템 콜이 발생하면 소프트웨어 인터럽트에 의해 커널 내부의 핸들러가 호출되고, 이후에는 시스템 콜 파라미터에 따라서 세부적인 각각의 서비스 루틴이 수행된다. 그러나 마이크로커널 기반의 구조에서는 시스템 콜 핸들러에 해당하는 루틴이 유저 레벨에서 수행되는 서버에 존재하기 때문에 IPC 를 이용해야 한다. 이러한 구조에서 응용프로그램과 UNIX 서버는 모두 유저 레벨의 프로세스이며, 응용프로그램이 UNIX 서버에게 시스템 콜을 요청하려면, 내부적으로는 모두 마이크로커널이 중재하는 IPC 메커니즘으로 변환이 되어야 한다. 이것으로 응용프로그램의 투명성(transparency)을 보장할 수 있다.

응용프로그램에서 시스템 콜을 libc 등의 라이브러리를 통해서 호출하면 내부적으로 소프트웨어 인터럽트를 발생시킨다. 그러나 시스템 콜에 해당하는 핸들러는 UNIX 서버 내에 존재하므로 일종의 *RPC(Remote Procedure Call)*가 이루어져야 한다. 다음 그림 3 은 이러한 상황을 보여주고 있다.

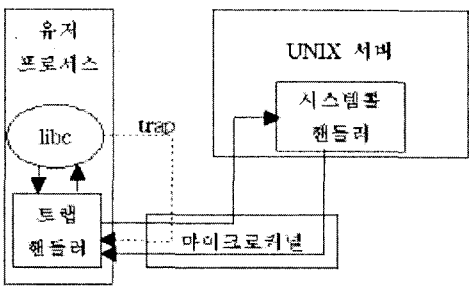

그림 3 마이크로커널 기반의 시스템 콜

소프트웨어 인터럽트 즉 트랩(trap)이 발생하면, 마이크로커널이 제어권을 가진다. 마이크로커널은 각 유저 프로세스의 주소 공간에 사상된 트랩 핸들러를 호출한다. 트랩 핸들러는 UNIX 서버의 시스템 콜 핸들러를 호출하기 위해서 IPC 메시지로 트랩의 내용을 변환하고 응답을 기다린다. 이후에, UNIX 서버로부터 응답을 받게 되면 트랩 핸들러에서 libc 로 복귀하여 계속 수행한다. 위와 같은 방법이 Mach 나 L4Linux 같은 마이크로커널에서 트램폴린(trampoline)[7]이라는 방법을 통해서 사용된다.

4. 비동기 IPC

시스템 콜 메커니즘에 관련된 IPC 는 UNIX 서버로부터의 응답이 있을 때까지 기다려야 하는 동기 IPC 메커니즘을 사용한다. 이러한 동기 IPC 는 메일 박스나 세마포어 등을 이용하게 된다. 하지만 보안 컴포넌트에서 보안 모니터로 메시지를 전달할 때는 불필요한 지연시간을 없애기 위해서 메시지 큐를 이용한 비동기 IPC 를 이용해야 한다. 이런 방법을 사용함으로 시스템 콜 모니터링에 따른 오버헤드를 감소시킬 수 있다.

4. 비교 및 분석

라이브러리 기반 접근 방식은 래퍼 루틴을 거치지 않고도 int 0x80 등을 이용하여 시스템 콜을 직접 호출할 수 있기 때문에, 침입 탐지 및 제한 시스템 같은 보안에 관련된 응용프로그램에서 쓰기에 적합하지 않다. 이를 극복한 커널 기반 접근 방식은 모든 확장코드가 커널 안에 들어가야 하므로, 커널의 많은 수정이 필요하다. 또한 확장 코드를 쉽게 고칠 수 없기 때문에 확장성과 유연성이 떨어진다. 또한 확장 코드가 커널 레벨에서 동작하므로, 잘못 작성된 코드는 시스템 전체의 지장을 줄 수 있다. 유저레벨 프로세스 기반 방식은 위의 두 가지 방법의 장점을 수용하였지만, 시스템 콜 인터셉션을 위한 부가적인 오버헤드가 존재한다. 이는 ptrace 나 strace 와 같은 별도의 시스템 콜을 사용하므로 유저 레벨 감시 시스템과 감시 대상 프로세스 간에 문맥 전환이 발생하기 때문이다.

본 논문에서 제시하는 방법은 유저 레벨의 보안 모니터 서버를 이용하므로 유저레벨 프로세스 기반 방식의 장점을 수용하고 있다. 또한 마이크로커널 내부의 IPC 를 이용하므로 별도의 시스템 콜이 필요 없고, 비동기 IPC 와 주기적인 보안 모니터 서버의 실행을 통해서 시스템 콜 감시에 따른 오버헤드를 최소화 하게 된다.

5. 결론

마이크로커널 기반의 시스템은 확장성이나 기능성, 유연성 등의 장점들을 가지고 있기 때문에 다양한 정책을 필요로 하는 정보보안 운영체제로써 적합한 구조이며 그 때문에 보안 운영체제 연구에 많이 사용되고 있다. 본 논문에서는 응용 프로그램에서 악의적인 공격을 시도하는 것을 탐지할 수 있는 메커니즘인 시스템 콜 인터포지션을 마이크로커널 기반의 시스템에서 적용하는 방법을 제안하였다. 본 논문이 제시하는 마이크로커널 기반 시스템 콜 인터포지션 방법은 기존의 방법들의 장점을 수용하고 있고 감시에 따른 오버헤드를 최소화하기 위해 노력하였다. 이 메커니즘을 이용하여 마이크로커널 기반의 침입 탐지 시스템을 구현하는데 큰 도움을 줄 수 있으며 그 성능도 향상할 수 있다.

참고문헌

[1] Andrew P. and Steven A., "Intrusion Detection via System Call Traces," IEEE Software, pp. 35-42, 1997.

[2] R. Spencer and S. Smalley et al, "The Flask Security Architecture: System Support for Diverse Security Policies," In Proceedings of the 8th USENIX Security Symposium, pp. 123-139, 1999.

[3] S. E. Minear, "Providing Policy Control Over Object Operations in a Mach Based System," In Proceedings of the 5th USENIX UNIX Security Symposium, pp. 141-156, 1995.

[4] M.Jones, "Interposition Agents: Transparently Interposing User Code at the System Interface," 14th Symposium on Operating Systems Principles, 1993

[5] T.Michem and R. Lu et al, "Using Kernel Hypervisors to Secure Applications," Annual Computer Security Application Conference, 1997

[6] K Jain and R Sekar, "User-level Infrastructure for System Call Interposition: A Platform for Intrusion Detection and Confinement," In ISOC Network and Distributed System Security, 2000.

[7] Hermann Härtig and Michael Hohmuth et al, "The Performance of μ-Kernel-Based Systems," 16th ACM Symposium on Operating Systems Principles, 1997

[8] Michael M. Swift and Brian N.Bershad et al, "Improving the Reliability of Commodity Operating Systems," 19th ACM Symposium on Operating Systems Principles, 2003