트로이목마 방지 샌드박스 메커니즘과 구현

김중완*, 김형찬*, 이동익*

*광주과학기술원, 정보통신공학과

A Sandbox Mechanism against Trojan Horses and Its Implementation

Joong-wan Kim*, Hyung-chan Kim*, Dong-ik Lee*

*Department of Information and Communications, K-JIST

요 약

본 논문에서는 자바의 샌드박스 개념을 이용하여 실행되는 프로그램에 제한을 가하고 모니터링을 수행함으로써 악의적인 행위를 방지하는 샌드박스 메커니즘을 제안한다. 또한 시스템 자원을 논리적 자원과 물리적 자원으로 분리하여 트로이목마 프로그램의 자원에 대한 불법적인 접근을 감시하여 기밀성(Confidentiality)과 무결성(Integrity)을 보장할 뿐만 아니라 물리적인 자원 사용량을 제한하여 DOS 공격에 대한 방지와 같이 가용성(Availability)까지 고려하는 샌드박스 메커니즘을 제안하고, 리눅스 커널 2.4.19에 샌드박스 메커니즘을 구현한다.

I. 서 론

인터넷의 발달과 FSF(Free Software Foundation)의 영향으로 소프트웨어의 자유로운 배포와 이용이 우리사회에 널리 퍼지고 있다. 리눅스는 그 제작동기에서부터 이용자로 하여금 커널 및 응용 소프트웨어 모듈을 자유롭게 배포하여 사용하게 하고 있다. 그러나 이와 같은 당초의 의도와는 달리, 인터넷에 배포된 소프트웨어에 트로이목마(Trojan horse)와 같은 악성코드를 삽입하여 침입을 시도하거나 피해를 주는 사례가 점점 증가하고 있다.

트로이목마 프로그램은 정상적인 기능을 하는 프로그램에 숨어서 의도하지 않은 악의적인 행위를 수행하는 악성코드를 말한다[1]. 특히 여러 악성코드 중에서 트로이목마 프로그램은 발견 및 대처가 어려워서 많은 연구가 이루어지지 못했다. 트로이목마 프로그램은 정상적인 프로그램으로 위장하여 수행되기 때문에 발견이 쉽지 않고[2], 프로그램 수행자의 권한을 불법으로 위임받아 수행되기 때문에 프로그램 수행자가 할 수 있는 모든 악의적인 행위를 할 수 있기 때문에 해킹에 준하는 매우 위험한 공격으로 간주되고 있다.

이러한 악성코드를 방지하는 기법으로 자바의 샌드박스(Sandbox)[3]의 개념을 이용하여 프로그램 실행 시 제한을 가하거나 모니터링을 함으로써 악성행위를 방지하는 기법이 최근 들어 많이 사용되고 있다. 그러나 기존의 샌드박스를 이용한 기법들은 파일시스템과 네트웍에 관한 제한에 한정되어 있고, 허가된 자원을 무한하게 할당하여 시스템의 정상적인 이용을 막는 DOS(Denial of Service) 공격에 대한 방지책이 없었다.

본 논문에서는 시스템 자원을 논리적인 자원과 물리적인 자원으로 분리하여 제한을 가하고 모니터링함으로써 기존 시스템의 문제점을 해결하고자 한다. 다시 말하면, 논리적인 자원에 대한 제한과 모니터링으로 시스템에 대한 접근권한을 통제할 수 있고, 물리적인 자원에 대한 제한과 모니터링으로 자원사용량을 통제하여 DOS 공격에 대한 방지가 가능하다.

시스템의 커널레벨에서 발생하는 시스템 콜을 변경하여 자원에 대한 논리적 접근권한을 검사하여 시스템 콜의 허용과 금지여부를 가려내서 비밀성(Confidentiality)과 무결성(Integrity)을 보장하고, 물리적인 자원에 대한 사용량을 검사하여 가용성(Availability)을 보장하는 샌드박스 메커니즘을 리눅스 커널 2.4.19에 구현함으로써 외부에서 유입되는 신뢰성 없는 어플리케이션에 대한 트로이목마와 같은 악성코드로부터의 악성행위를 방지하여 안전한 컴퓨팅 환경을 제공하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 악성코드를 방지하는 기존의 연구에 대하여 소개하고, 3장에서는 새로운 트로이목마를 방지하는 샌드박스 메커니즘의 일반적인 구조에 대하여 설명하고, 4장에서는 논리적인 자원에 대한 제어와 물리적인 자원에 대한 제어에 대해서 설명하고, 5장에서는 샌드박스 메커니즘의 구현과 동작에 대해서 설명한다. 마지막으로 6장에서는 결론을 맺고 향후연구에 대해서 기술한다.

Ⅱ. 관련연구

1. 인증서를 이용하는 방법

신뢰할 수 있는 제3자가 프로그램의 정보와 해쉬값을 포함하는 인증서를 발행하여 프로그램과 같이 배포하게 하면, 사용자는 이 인증서를 검증하여 프로그램의 작성자와 프로그램의 안전성에 관한 정보 그리고 이들의 무결성에 대한 증거정보를 확인하는 것이다. 이러한 방법은 그 파일들이 무단으로 변경되지 않았다는 것을 암호학의 해쉬함수와 전자서명의 강력함에 그 근거를 두고 보증하고 있다[4]. 이러한 방법은 현재 일부 상업적인 프로그램에서 사용되고 있기는 하나 리눅스와 같이 GNU 라이센스에 의해 누구나 프로그램을 수정하고 배포할 수 있는 환경에서는 적절하지 못한 방법이다.

2. 증거코드 삽입(Proof Carrying Code)

증거코드 삽입은 호스트 시스템이 프로그램을 실행해도 안전한지에 대한 증거정보를 사용하여 안전성을 보장하게 하는 방법이다. 이 방법을 사용하기 위해서는 프로그램 작성자는 프로그램의 요구조건에 만족하는 정형적 증거를 프로그램에 부착하여야 하며, 사용자가 그 프로그램이 안전한지를 검증할 수 있도록 해준다[5,6]. 이와 비슷한 또 다른 접근은 정형적 증거 대신에 시스템 자원에 대한 접근리스트(access list)를 사용하여 그 프로그램을 시스템에서 수용할 수 있을지를 결정하게 된다[7].

3. 샌드박스 어플리케이션

악성코드에 대한 연구는 자바 애플릿의 보안을 위해 사용했던 샌드박스의 개념을 이용하여 제한된 실행환경을 사용하기 시작했다. 프로그램의 악의적인 행동을 방지하기 위해서 보안정책에 따라 제한된 실행환경에서 프로그램을 실행함으로써 프로그램의 악의적인 행동을 방지한다. 샌드박스를 이용한 방법중에서 가장 대표적인 연구는 Janus [8,9] 시스템이다. Janus는 ptrace 시스템 콜이나 /proc 파일 시스템을 사용하여 신뢰성 없는 프로그램으로부터 발생하는 시스템 콜을 필터링하여 악의적인 행동을 방지한다.

그러나 Janus는 몇 가지 단점을 가지고 있다. 첫째는 파일시스템과 네트웍에 대한 제어에 한정되어 있고, 두 번째로는 허가된 자원을 무한하게 할당하여 시스템의 정상적 정상적인 동작을 방해하는 DOS 공격을 방지할 방법이 없다는 점이다. 세 번째는 사용자 레벨의 응용프로그램이기 때문에 프로그램이 실행되지 않았거나 시스템 공격자가 강제적으로 프로그램을 종료하게 함으로써 바이패싱(bypassing)이 가능하다. 네 번째는 운영체제에 따라서 포팅(porting)이 어렵다는 점이다. 마지막으로는 시스템 콜의 파라미터로 악의

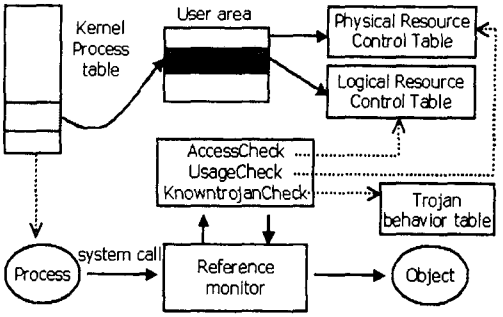

그림 1 샌드박스 메커니즘의 전체적인 구조

성을 판단하지 않고 시스템 콜 자체만으로 악의성을 판단한다는 점이다.

Ⅲ.샌드박스 메커니즘의 구조

트로이목마 프로그램은 사용자가 알지 못하게 비밀스럽고 악의적인 행동을 수행한다. 또한 사용자가 할 수 있는 모든 일을 수행할 수도 있다. 이러한 트로이목마 프로그램의 악의적인 행동은 바로 트로이목마 프로그램이 사용자의 권한을 가지고 사용자가 알지 못하는 사이에 불법적으로 시스템 자원을 접근하여 행한 결과다. 이것은 프로그램의 실행에 필요한 권한보다도 많은 권한을 부여했기 때문에 발생하는 결과이다. 사용자에게 부여된 권한이 실행시키고자 하는 프로그램으로 위임되고, 또 그 프로그램에 삽입되어 있는 트로이목마 프로그램에 계속해서 권한이 위임되기 때문이다. 따라서 최소 권한의 법칙(the principle of least privilege)에 따라 자바 애플릿의 보안을 위한 샌드박스의 개념을 이용하여 시스템 자원의 접근을 제한하고 감시하는 방법을 이용하면 트로이목마 프로그램의 악성행위를 방지할 수 있다. 최소 권한의 법칙이란 시스템이 안전하기 위해서는 시스템의 실행에 필요한 최소한의 객체에 대한 접근권한을 허용해야 한다는 원칙이다[1].

본 논문에서는 시스템 자원을 논리적인 자원과 물리적인 자원으로 분리하여 제한을 가하고 모니터링함으로써 기존 시스템의 문제점을 해결하고자 한다.

이 샌드박스 메커니즘의 동작은 설정된 보안정책에 따라 결정되며, 프로그램마다 사용하는 권한과 자원의 양이 다르기 때문에 프로그램 각각의 보안정책을 설정하여 프로그램이 동작하는데 필요한 권한을 설정해 주어야 프로그램이 정상적으로 동작할 수 있다. 샌드박스 메커니즘의 보안정책은 세 부분으로 구성되는데, 프로그램의 실행권한을 설정하는 논리적 자원제어 부분과 프로그램의 자원사용을 제어하는 물리적 자원 제어 부분과 트로이목마 프로그램의 행동 특징을 기술한 부분으로 구성된다. 먼저 트로이목마 프로그램의 행동 특성은 지금까지 리눅스에서 문제를 일으켰던 트로이목마 프로그램들의 특성들을 분석하여 알려진 트로이목마 프로그램들을 탐지하기 위한 것이다.

파일, 디렉토리, 네트웍 포트, 메모리 페이지, 프린터의 스풀과 같은 논리적인 자원에 대한 제어는 실제 프로그램이 가지는 접근권한을 제어할 수 있다. 따라서 접근이 정당하고 필요한 경우에만 접근을 허용하게 한다면 기밀성(confidentiality)과 무결성(integrity)을 보장하고, 알려지지 않은 일반적인 악성행위를 방지할 수 있다. 또한 물리적인 메모리, 디스크, CPU 사용, I/O 사용과 같은 물리적인 자원에 대한 제어는 프로그램이 사용할 수 있는 자원의 양을 제한하여 허가된 자원을 무한하게 할당하여 시스템의 정상적인 동작을 막는 DOS 공격을 방지하여 가용성(availability)을 높여줄 수 있도록 설계하였다.

샌드박스 메커니즘의 일반적인 구조는 그림1과 같이 구성된다. 먼저 프로그램이 시작되어 메모리에 로드 될 때, 프로그램마다 각각의 설정된 보안정책을 동시에 불러들여 테이블로 만들어 놓는다. 이 보안정책은 각각의 프로세스에서 참조가 가능하도록 커널내의 프로세스 영역의 사용자 영역에 보안정책을 참조할 수 있도록 연결시키고, 프로세스가 참조모니터를 통하여 자원을 요청할 때 발생하는 시스템 콜을 변경하여 접근권한검사(AccessCheck), 자원사용량검사(UsageCheck), 알려진 트로이목마 프로그램의 일반적인 특징검사(KnowntrojanCheck)를 통하여 보안정책에 설정된 권한허용과 자원사용량을 위반하는지 검사하고 감시하게 된다. 접근권한과 자원사용이 정당하면 참조모니터를 통하여 시스템 자원의 요청을 사용할 수 있도록 허가하고, 이에 위반되면 자원할당에 대한 요청은 금지시켜서 트로이목마와 같은 악성코드의 악성행위를 방지할 수 있도록 구성하였다.

Ⅳ. 시스템 자원에 대한 제어

1. 논리적인 자원에 대한 제어

파일, 디렉토리, 네트웍 포트, 메모리 페이지, 프린터의 스풀과 같은 논리적인 자원에 대한 제어는 논리적인 자원에 대한 접근권한을 제한하거나 감시할 수 있다. 프로그램이 메모리에 로드되어 프로세스로서 참조모니터를 통하여 시스템 자원을 요청할 때 발생하는 시스템 콜을 변경하여 시스템 콜이 발생할 때 논리적인 자원에 대한 접근권한 검사를 수행하도록 하였다.

수정된 시스템 콜은 그림2와 같이 수정하여 시스템 콜이 호출되면 먼저 논리적인 자원에 대한 접근권한을 검사하는 AccessCheck() 모듈을 불러서 시스템 콜 이름과 시스템 콜의 인자값을 메모리에 로드된 논리적 자원에 대한 접근권한에 관한 보안정책과 비교하여 정책에 위반되는지를 검사한다. 보안정책에 위배되지 않는 시스템 콜이라면 정상적인 open 시스템 콜을 호출하여 진행되도록 수정하였다.

시스템 콜을 수정하여 시스템 콜의 호출 시에 불리워져서 논리적인 자원에 대한 접근권한을 검사하는 AccessCheck() 모듈은 그림3와 같이 의사코드로 알기 쉽게 표현하였다. 먼저 시스템 콜의

```
Open() {
    Call AccessCheck();
    If allowed, call the open system call;
    return; }
```

그림 2 수정된 open 시스템 콜의 구조

이름과 시스템 콜의 인자값을 기준으로 하여 요청한 자원에 대한 접근권한의 VerifyOption이 Super_Allow이면 바로 허용하고, Super_Deny이면 바로 금지시키도록 한다. 그리고 Verify_Allow에 해당하면 parameter를 검사하여 리스트에 있다면 허용하고, Verify_Deny에 해당되면 parameter를 검사하여 리스트에 있다면 금지시키도록 구성하였다.

2. 물리적인 자원에 대한 제어

물리적인 메모리, 디스크, CPU 사용, I/O 사용과 같은 물리적인 자원에 대한 제어는 프로세스

```
AccessCheck() {
    if (VerifyOption == Super_Allow) return YES;
    if (VerifyOption == Super_Deny) return NO;
    if found in allowable parameter range then return YES;
    if found in non-allowable parameter range then return NO;
    }
```

그림 3 AccessCheck 모듈의 의사코드

가 요청한 시스템 자원에 대한 사용량을 제한하여 허가받은 시스템 자원이라 할지라도 무한하게 할당하여 시스템의 정상적인 운영을 방해하는 DOS 공격을 방지하여 가용성(availability)을 높여줄 수 있는 방법이다. 물리적인 자원에 대한 보안정책도 역시 보안정책파일에 기술된 보안정책을 프로그램이 메모리에 로드 될 때 같이 메모리에 로드되어 테이블로 구성된다.

물리적인 자원의 사용량은 검사하는 모듈인 UsageCheck는 그림3과 같이 알기 쉽게 의사코드로 표현하였다. 먼저 요청된 자원의 양(REQAMT)과 현재 사용량을 계산하여 최대 자원 사용량(MaxUsage)를 넘으면 요청된 자원은 할당을 금지시킨다. 최대 사용량을 넘지 않는다면 새로운 자원 소비율을 계산한다. 새롭게 계산된 자원 소비율도 최대 자원 소비율(MaxConsmpRate)을 넘지 않는다면, 현재 자원 사용량(CurrentUsage)을 업데이트 한다. 최대 자원 소비율을 넘는다면 현재 소비되고 있는 자원의 소비율이 최대 소비율을 넘지 않을 대략의 시간을 계산하여 예상되는 최소 지체시간을 리턴하고 자원 할당은 금지하고 종료된다.

```
UsageCheck(REQAMT) {
        If ((REQAMT + CurrUsage) > MaxUsage))
            return(FAILURE);
        compute NewConsumptionRate();
        If (NewConsmpRate > MaxConsmpRate) {
            compute MinDelay before consumption;
            return(MinDelay); }
        UpdateCurrentUsage();
        return(SUCCESS);
        }
```

그림 4 UsageCheck 모듈의 의사코드

V. 구현과 동작

외부에서 유입되는 신뢰성 없는 어플리케이션의 악성행동을 방지하는 샌드박스 메커니즘은 리눅스 커널 2.4.19에 구현되었다. 본 논문에서 제안된 샌드박스 메커니즘은 커널레벨에서 동작하도록 설계되었고, 커널의 수정과 커널 모듈 프로그래밍을 통하여 구현되었다.

먼저 논리적인 자원에 대한 제어는 프로세스의 정보영역인 task_struct 구조체에 사용자 영역을 추가하여 샌드박스 메커니즘에서 필요한 정보를 유지하도록 변경하였다. 그리고 프로그램 파일에 고유의 SID(Security IDentifier)를 갖도록 하여 프로세스와 그에 따라 발생하는 시스템 콜을 구별하도록 하였다. 이에 따라 inode에 SID 값을 읽고 쓸 수 있는 시스템 콜을 추가하고, 프로세스의 시작(execve)과 종료(exit) 시에 보안정책을 초기화하고 제거할 수 있도록 process.c 파일을 수정하여 커널을 재 컴파일 하였다. 그리고 실제 시스템자원에 대한 제어를 담당하는 AccessCheck 모듈과 UsageCheck 모듈 그리고 KnownTrojanCheck 모듈은 커널 모듈 프로그램으로 구현하였고, 실제 보안정책을 입력하는 툴을 만들어서 그림5와 같이 보안정책의 입력을 용이하게 하였다.

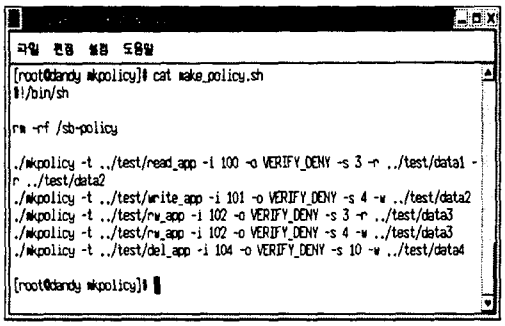

그림 5 보안정책 설정

물리적인 자원에 대한 정보는 리눅스 시스템의 /proc 파일 시스템을 통하여 정보를 얻도록 하였다. /proc 파일 시스템은 실제로 존재하지 않는 가상의 파일시스템으로서 메모리에 존재하는 시스템 정보를 사용자가 보고자 요청하는 경우에 발생하는 커널상의 시스템 정보 매개체이다.

메모리에 존재하는 시스템정보인 /proc 파일 시스템은 파일로 쓰여지기 전에는 커널상의 데이터구조로 표현되어 있으므로, 커널 모듈 프로그램에서는 /proc 파일을 직접 읽지 않고 커널상의 데이터를 참조할 수 있다. 따라서 UsageCheck 모듈은 이러한 /proc 파일 시스템을 통하여 현재의 자원 사용량과 과도한 자원요청이 없는지를 확인한다.

그림5의 보안정책 입력에서 rw_app라는 프로그램에 data3이라는 파일에 대한 읽기와 쓰기를 금지시키는 정책을 주었다. 그림6은 간단한 read/write를 시험할 수 있는 프로그램인 rw_app 프로그램이 data3 파일에 대한 read와 write 동작

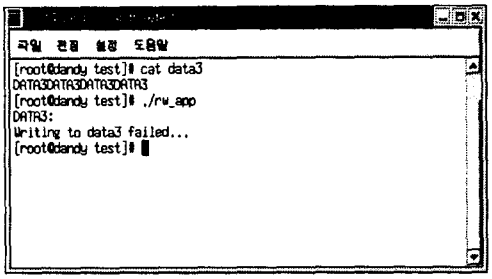

그림 6 rw_app의 read/write 실패

의 실패 모습을 보여준다.

VI. 결론 및 향후연구

본 논문에서는 악성코드를 포함하고 있을지도 모르는 외부에서 유입된 프로그램에 대해서 실행되는데 필요한 최소한의 실행권한과 자원 사용량을 할당하는 방식으로 제한하여, 트로이목마와 같은 악성코드가 실행되지 못하도록 안전한 실행환경을 만들 수 있는 샌드박스 메커니즘을 제안하고 리눅스 커널 2.4.19에 구현하였다.

본 논문에서 제안한 샌드박스 메커니즘은 논리적인 모든 자원에 대한 접근권한을 명시하여 제한하고 감시할 수 있도록 구성되었고, 물리적인 자원의 사용량을 제한하고 감시하여 허가받은 시스템 자원이라고 할지라도 사용량의 제한하고 감시하여 DOS 공격을 방지할 수 있기 때문에, 기밀성(confidentiality), 무결성(integrity), 가용성(availability)을 모두 고려하는 샌드박스 메커니즘이라고 할 수 있다. 또한 기존에 알려진 트로이목마 프로그램의 탐지뿐만 아니라 알려지지 않은 악성코드의 악성행위도 방지할 수 있는 방법이다. 프로세스의 어떠한 동작도 이 시스템 콜과 참조모니터에 객체를 요청하여 할당받는 형식을 벗어날 수 없기 때문에 바이패싱이 불가능하고, 시

스템 콜의 검사옵션에 따라 무조건적인 허용과 금지하는 시스템 콜 정책과 시스템콜의 인자 값에 따라 허용과 금지를 판별하는 효율적인 방법을 사용하였다.

현재 보안정책의 정형화에 대한 추가연구가 필요하며, 프로그램을 설치하는 사용자마다 다르게 설정할 수 있는 보안정책에 대한 일반화에 관한 향후 연구가 필요하다고 생각된다.

참고문헌

[1] Charles P. Pfleeger, Security in Computing, Prentice Hall, 1997.

[2] H. Thimbleby, S. Anderson, and P. Cairns, A Framework for Modeling Trojans and Computer Virus Infection, Computer Journal, 41(7):444-458, 1999.

[3] Li Gong, Marianne Mueller, et al., Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2, Proc. of the USENIX Symposium on Internet Technologies and Systems, 1997.

[4] Aviel D. Rubin, Trusted Distribution of Software Over the Internet, Proc. of the Internet Society Symposium on Network and Distributed System Security, 1995.

[5] George C. Necula, Proof-Carrying Code, POPL、97, 1997.

[6] Grigore Rosu and Nathan Segerlind, Proofs of Safety for Untrusted Code, UCSD Technical Report CS1999-633, 1999.

[7] Eun-Sun Cho, Sunho .Hong, Sechang Oh, Hong-Jin Yeh, Manpyo Hong, Cheol-Won Lee, Hyundong Park, and Chun-Sik Park, SKETHIC: Secure Kernel Extension against Trojan Horses with Information-Carrying Code, ACISP 2001:177-189, 2001.

[8] Ian Goldberg, David Wagner, Randi Thomas and Eric A. Brewer, A Secure Environment for Untrusted Helper Applications, Proc. of the 6th USENIX Security Symposium, 1996.

[9] David Wager, Janus: An Approach for Confinement of Untrusted Applications, UC Berkely, 1999.