리눅스 환경에서 보안 모델을 위한 객체 분류 방법

임종혁*, 박재철*, 김동국**, 노봉남*

*전남대학교 정보보호협동과정

** 전남대학교 전자컴퓨터정보통신공학부

Object Classification Method for Security Model Based on Linux System

JongHyuk Im*, JaeChul Park*, DongKook Kim**, BongNam Noh*

*Interdisciplinary Program of Information Security, Chonnam National University.

**Dept. of Electronics Computer & Information Engineering, Chonnam Natiomal University.

요 약

최근 활발히 개발 중인 보안운영체제의 핵심인 보안커널(security kernel)은 참조모니터(reference monitor)에서 주체(subject)가 객체(object)에 대한 실행(action) 권한을 판단함으로써 접근 제어를 실행한다. 보안운영체제의 대표적인 접근제어모델에는 다중레벨접근제어(MLS: Multi Level Security)모델과 역할기반접근제어(RBAC: Role Based Access Control) 모델 등이 있다. 리눅스 시스템에서 이러한 접근제어모델을 적용하기 위해서 접근 대상이 되는 객체들의 효과적인 분류가 요구된다. 본 논문에서는 리눅스 환경에서 효과적인 접근제어모델을 적용하기 위하여 객체들을 객체 클래스(class)와 유형(type)을 기준으로 분류 하였다.

I. 서론

안전한 정보 공유와 보호를 위해 방화벽, 침입탐지 시스템 및 암호화 메커니즘 등 많은 보안 기술이 개발되었지만 이러한 보안기술은 애플리케이션 수준에서 운용되기 때문에 잠재적인 보안 취약점과 내부자의 침입과 권한 남용 및 오용에 대한 대응이 어렵다. 이러한 근본적인 문제를 해결하기 위해서 새로운 보안 패러다임의 변화와 1)더불어 정책 구현과 실현을 위한 보안운영체제의 중요성이 대두되고 있다[1].

보안운영체제의 접근제어는 인가된 사용자는 허가된 작업을 수행 할 수 있고 그렇지 않은 사용자는 할 수 없다는 보증을 원한다. 접근제어는 기본적으로 '주체(Subject)'의 '객체(Object)'에 대한 실행(Action)' 권한을 판단함으로써 접근제어를 실행하는데, 이를 위하여 많은 시스템들이 MLS나 RBAC과 같은 접근제어 모델을 이용해 구현을 하고 있다. 결국 접근제어라는 것은 주체의 행위 대상인 객체에 대한 접근이 적절한지를 결정하는 것으로, 효율적인 접근제어를 위해 이들 객체에 대한 분류가 필요하다[2, 3]. 본 논문은 기존 보안운영체제에 적용된 객체 분류와 그 문제점을 살펴보고, 접근제어의 유연성과 다양한 접근제어를 적용할 수 있는 객체분류 방법을 제안한다.

* 본 연구는 정보통신부 대학 IT 연구센터 육성, 지원사업의 연구결과로 수행되었습니다.

II. 관련연구

2.1 SELinux(Security Enhanced Linux)

SELinux는 NSA(National Security Agency)의 주도하에 여러 연구소에 의해 구현된 보안 운영체제이다. 기본적으로 Flask 구조에 바탕을 두고 있으며, 자신들이 개발한 TE(Type Enforcement) 모델을 기반으로 다양한 접근제어 모델을 사용 할 수 있는 보안운영체제를 만들었다[4]. 특히, 전통적인 강제적 접근제어의 문제점을 극복하기 위해 보안 정책 집단과 그것을 행하는 메커니즘을 분리시켜서 다양한 강제적 접근제어 정책을 가능하게 하고 유연하고 사용하기 편리한 방법을 제공한다[5].

2.1.1 SELinux의 객체 분류

SELinux는 LSM(Linux Security Module)을 기반으로 하여 리눅스 시스템에 있는 객체들을 각각의 클래스들로 보안 객체 클래스를 만들어 냈다[6]. 디렉터리와 디바이스, 그리고 메모리를 File 클래스로 묶고 그 외에 공통적인 부분들을 묶어 Interprocess Communication 클래스, Network 클래스, Object 클래스, System 클래스의 다섯 개 클래스로 분류해 놓았다. 분류해 놓은 객체에 대한 행위들에 대해서는 Append, Create, Execute, Get attribute, I/O control, Link, Lock, Read, Rename, Unlink, Write들을 기본 행위로 지정해 놓고, 이들 각각에 대해서 다시 객체 클래스를 분류했다. 즉, 객체 클래스 중심이 아닌 오퍼레이션 중심으로 객체 클래스들을 분류하였다[5]. 그러나 SELinux는 LSM을 사용하여 구현하였기 때문에 LSM에 등록되어 있지 않은 시스템 콜들에 대해서는 접근제어를 할 수 없다는 단점이 있다. 또한 자신들이 개발한 TE 모델을 이용하여 다양한 접근제어 모델을 사용할 수 있었지만, 현재 리눅스 시스템에서 사용하고 있는 DAC(discretionary Access Control) 위에 TE 모델을 사용하고 있기 때문에, 의도되지 않은 접근이나 행위에 대해서 DAC 모델에서 먼저 접근제어가 일어 날 수 있다는 단점이 있다.

2.2 REMUS

REMUS는 시스템 콜을 모니터링 하기 위하여 처음에 개발 되었으며, OS Kernel 레벨에서 시스템 콜을 후킹 할 수 있는 간단한 메커니즘을 이용하였다. REMUS는 참조모니터의 구현으로 시스템 콜이 호출이 되었을 경우, 참조 모니터에서 이를 DB(Database)의 자료와 비교하여 승인 또는 승인거부를 해주게 된다[7].

2.2.1 REMUS의 객체 분류

REMUS의 특징은 기능별로 그룹을 나누고 위험도별로 따로 그룹을 나누어 놓았다. 이는 리눅스 시스템이 가지고 있는 각각의 오퍼레이션들이 가지고 있는 위험도에 따라 오퍼레이션을 관리 할 수 있다는 장점이 있다. REMUS는 기능별로 9개의 그룹으로 나누어 각각의 그룹을 로마 문자로 표현 하였다. 그룹Ⅰ은 파일 시스템과 디바이스, 그룹Ⅱ는 프로세스 관리, 그룹Ⅲ은 모듈 관리, 그룹Ⅳ는 메모리 관리, 그룹Ⅴ는 타임과 타이머를 그룹Ⅵ은 커뮤니케이션, 그룹Ⅶ은 시스템 정보, 그룹Ⅷ은 예비, 그룹Ⅸ 구현되지 않은 것들 이렇게 9개의 그룹으로 나누고 4단계의 위험 레벨을 나눈 후 위험도 별로 오퍼레이션들을 분류하였다. 레벨은 숫자가 낮을수록 위험한 단계고, 숫자가 높을수록 안전한 단계이다. 레벨 1은 시스템의 전 제어권을 획득할 수 있는 단계, 레벨 2는 서비스 거부 공격을 사용할 수 있는 단계, 레벨 3은 프로세스를 파괴 할 수 있는 단계, 그리고 마지막 레벨 4는 무해한 단계이다. 각각의 오퍼레이션들은 그룹별로 나누어지고, 그 나누어진 오퍼레이션들은 다시 위험도 레벨별로 다시 나누어진다. 이렇듯, REMUS는 위험도에 따라 시스템 콜을 구분하고 접근제어를 실행한다. 이는 리눅스 시스템 상에서의 시스템 콜들의 위험한 정도를 4단계로 분리하여 판단할 수 있게 되므로, 훨씬 더 효율적으로 접근제어를 실행 할 수 있으나, 이들이 나누어 놓은 위험도에 관한 정확한 기준이 모호하다. 때문에 오히려 접근제어 정책에서 더욱 혼란을 가져 올 가능성이 충분히 있다.

III. 제안하는 객체 분류 방법

유닉스 계열 시스템들은 대부분 객체들을 파일 형태로 표현하고 관리한다. 그러나 모든 객체들을 파일 형태로 관리하게 되면 접근제어 서비스를 위한 자료구조 및 관리가 복잡하게 되는 문제점이 발생하므로 공통된 특성을 갖는 객체들을 클래스별로 분류하여 관리하면 보다 효율적인 접근제어를 할 수 있다.

3.1. 접근객체의 분류

본 논문에서는 리눅스 시스템의 자원 특성에 따라 표 1과 같이 FILE, IPC, SYSTEM, PROCESS, NETWORK, USER, UNCLASSIFIED의 7개 클래스로 분류하였다.

[표 1] 접근객체의 분류

클래스	유 형	설 명
FILE	File	아이노드 번호, 디바이스로 식별
	Dir	아이노드 번호, 디바이스로 식별.
	File System	파일 시스템 형식으로 식별
	Device	디바이스 타입, 디바이스 메이저, 마이너 번호로 식별
IPC	IPC	세마포어, 메시지, 공유 메모리, 소켓 , FiFo
SYSTEM	System	시스템과 연관된 객체
PROCESS	Process	pid로 식별.
	Signal	시그널 번호로 식별
NETWORK	Socket	소켓 관련 시스템콜
	IP	
	Port	
	Protocol	
USER	User	uid로 구별
UNCLASSIFIED	Unclassified	분류되지 않는 객체

접근제어를 수행할 때 접근주체의 접근권한 검사가 필요하거나 앞의 클래스 분류에 속하지 않는 객체들을 관리하기 위해 미분류 클래스를 추가하였다. 또한, 각각의 접근객체 클래스들은 한개 이상의 하위클래스인 객체 유형들로 구성된다. 메모리 객체는 리눅스 운영체제에서 가상 메모리 방식으로 관리하므로 일반 사용자의 운영체제가 허용하지 않는 방법으로 다른 사용자의 메모리 영역을 접근할 수 없다. 즉, 운영체제가 직접 메모리 관리를 하므로 운영체제의 기능을 제한하는 특별한 접근권한을 주체에게 부여할 필요성이 없다. 따라서 본 논문에서는 메모리객체에 대한 접근권한은 파일 객체나 프로세스 객체 등에서 간접적으로 관리한다.

3.2. 오퍼레이션 분류

본 논문에서는 클래스별 유형을 지정해 줌으로써, 기능별로 세세하게 분류 할 수 있었고, 서로 간에 의존성을 최소화 시킬 수 있었다. 오퍼레이션을 분류 할 때에는 리눅스에서 있는 모든 객체들이 파일로 다뤄지기 때문에 오퍼레이션의 분류가 애매하다. 따라서 더 강한 성향을 띄는 쪽으로 오퍼레이션을 분류하였다. 예를 들어, socket_file이 있다면 이것은 FILE 속성과 NETWORK 속성 양쪽 모두의 속성을 가지고 있을 것이다. 이런 것들은 NETWORK 유형으로 분류 하였다.

FILE 클래스의 File 유형은 파일과 직접적인 관련이 있는 것들만을 포함 시켰다. read나 write의 경우 FILE 클래스뿐만 아니라 다른 클래스들에도 중복으로 지정 될 수 있다. 네트워크 클래스에서의 read, write는 오퍼레이션은 같아도 명시적인 기능은 다르기 때문에 중복 지정해 놓았다.

IPC는 그 자체만으로도 보안과 관련하여 직접적인 위협이 많으므로 클래스를 분리 하였다. 공유 메모리나 세마포어 같은 오퍼레이션들은 공격자에게 넘어가면 위험하므로, 이것들을 관리하기 위한 IPC 클래스를 만들어 놓은 것이다.

네트워크 클래스는 Socket, IP, Port, Protocol

의 4가지의 유형을 가지고 있다. 기본적인 오퍼레이션들은 Socket 유형으로 분류 될 수 있겠지만, 관리의 효율성을 위하여 세분화 시켰다.

USER 클래스는 사용자와 관련된 오퍼레이션 즉, add/delete user 와 같은 오퍼레이션들이고 UNCLASSIFIED 클래스는 객체가 없는 정책을 설정하거나, 구현이 되어 있지 않은 시스템 콜을 위하여 만든 클래스이다. 본 논문에서는 구현되어 있지 않은 시스템 콜이라도 모두 관리 대상에 포함을 시켰고 자세한 내용은 부록에 첨부하였다.

IV. 결론

객체를 효율적으로 분류 시키는 작업은 정확한 접근통제를 제공하기 위해 꼭 필요한 작업이다. 객체 분류가 잘못된다면, 보안 모델의 접근제어 결정에 큰 영향을 미치기 때문이다. 본 논문에서는 효율적인 접근제어를 위하여 리눅스 시스템의 객체들을 자원 특성에 따라 클래스별로 분류하였다.

본 논문에서 제시한 분류 방법을 통하여 SELinux의 객체 분류 방법에 따른 의도되지 않은 접근이나 행위에 대한 접근제어의 충돌과 LSM 사용으로 인한 구현의 한계를 극복할 수 있다. 또한, REMUS의 단점인 위험도 기준의 모호함과 접근제어 정책의 혼란을 피할 수 있어 기존의 객체 분류의 복잡함을 줄이고 효율적인 접근제어 모델의 적용 가능성을 보였다. 제안한 객체 분류 방법은 동적인 접근통제 모델[8]을 운영체제에 구현하여 적용할 수 있었고, 기존 객체 분류 방법이 제공할 수 없었던 유연성을 제공하였다.

향후, 객체 분류를 기반으로 한 접근제어정책 모델과 그것을 적용한 보안운영체제의 성능 평가를 통하여 지금까지 나타나지 않은 문제점들을 보완하는 연구가 필요하다.

[참고문헌]

[1] 홍기융, 김재명, 홍기완. "Secure OS 보안정책 및 메커니즘". 정보보호학회지, 제 15권 제4호, Aug. 2003.

[2] Amon Ott, "The Rule Set Based Access Control Linux Kernel Security Extension", International Linux Kongress 2001.

[3] Leonard J. La Padula "Formal Modeling in a Generalized Framework For Access control", Third IEEE Conputer Security Foundations Workshop - CSFW'90, Franconia New Hampshire USA, 12-14 June 1990, Proceeding. IEEE Computer Society, Pages 100-109.

[4] Ray Spencer, Stephen Smalley, Peter Loscocco, Mike Hibler, David Anderson, Jay Lepreau, "The Flask Security Architecture: System Support for Diverse Security Policies", The Eighth USENIX Security Symposium, August 1999, pages 123-139.

[5] Bill McCarty, SELINUX NSAs open source security enhanced linux, O'Reilly.

[6] Philippe Biondi, "Kernel Level Security", 10th International Linux System Technology Conference, 26th September 2003.

[7] M. Bernaschi, E. Gabrielli, L. V. Mancini, "REMUS: A Security-Enhanced Operating System", ACM Transactions on Information and System Security, Vol. 5, No. 1, February 2002, Pages 36-61.

[8] 김정순, 김민수, 노봉남, "유연한 접근통제를 제공하는 보안 운영체제를 위한 접근통제 보안구조", 정보보호학회지, 2006.

[부록]

[표 2] 클래스별 오퍼레이션 분류

Class	Type	Operations
FILE	File (26)	Create, Delete, Open, Close, Read, Write, Execute, Truncate Rename, Modify MAC time, Change owner, group Get/Set status, Get/Set permission, Get/Set extended attribute Symbolic/Hard link, File control(fcntl), File lock/unlock Make node, Mmap, Munmap, Mprotect, Munprotect, Send file
	Dir (24)	Create, Delete, Open, Close, Read, Rename, Modify MAC time Change working directory, Change owner, group Change root directory, Get/Set status, Get/Set permission Create/Delete file, Create/Delete directory, Symbolic link/unlink Make node, Mount, Unmount
	File System (6)	Register, Unregister, Get filesystem statistics Change root(/) filesystem(pivot_root), Mount, Unmount
	Device (9)	Open, Close, Read, Write Ioctl, Register, Unregister(*), Mount, Unmount
IPC	IPC (10)	Create, Delete IPC(msg, shm, sem, fifo..), Read, Write Rename(FIFO), Get permission, Get status Control shared memory, Control message queue Control semaphore
SYSTEM	System (29)	Create/Init/Delete/Query module, Reboot(shutdown) Process trace, Sysctl, Syslog, Uname , Quota control, Quota on Adjust timex, Swap on/off vm_enough_memory, register_security, unregister_security Get/Set time, Get/Set time of day, Get/Set/Check capability Get rusage, Get kernel symbols Set resource limits(rlimit), Set hostname, Set domain name
PROCESS	Process (33)	Create, DeleteWait, Sleep(sleep, nanosleep), Fork, Execute Process trace, Switch process accounting Get/Set scheduler, Get/Set program scheduling priority Get/Set LDT(local descriptor table) Get/Set/Check capability, Get/Set interval timer Get/Set/Create/Delete per-process timer Change owner/group, Change working directory Change process priority(nice), Change root directory Change parent(reparent to init), Get scheduling parameter Get round-robin interval Set process execution domain(personality) Set resource limits(rlimit)
	Signal (11)	Signal return, Kill, alarm, Pause, Sigaction, Suspend, Pending Get/Set signal mask, Get/Set alternate signal stack content
NETWORK	Socket (11)	Create(socket), Close(shutdown), Read, Write, Bind, Listen Accept, Connect, GetPeername, Get/Set socket options
	IP (3)	Accept, Bind, Connect
	Port (2)	Bind, Connect
	Protocol (1)	Create(raw socket, TCP, UDP, ICMP, IGMP)
USER	User (4)	Add/Delete user, Switch user, Assign role
UNCLASSIFIED	Etc(**) (10)	Fast userspace locking system call, Remap file pages Set thread id address, Get/Set memory policy Get/Set TLS area, Add/Request/Control key