리눅스 운영체제에서 접근 및 행위제어를 위한 참조모니터 구현

김형찬*, 신 욱**, R. S. Ramakrishna*
*광주과학기술원 정보통신공학과
**University of Illinois at Urbana-Champaign 컴퓨터과학과
e-mail:{kimhc, rsr}@gist.ac.kr, wookshin@uiuc.edu

An Implementation of Reference Monitor for Access and Behavior Control in Linux

Hyung Chan Kim*, Wook Shin**, R. S. Ramakrishna*
*Dept. of Information & Communications,
Gwangju Institute of Science and Technology
**Dept. of Computer Science,
University of Illinois at Urbana-Champaign

요 약

운영체제에서 기존의 접근통제 서비스는 조직의 보안 요구사항을 반영하여 기밀성과 무결성 등을 지원하였다. 하지만 많은 경우의 프로그램 수행 시간 공격(Program Runtime Attack)들은 행위적인 의미를 수반하며, 이는 개개의 접근 인스턴스가 아닌, 접근 인스턴스의 연속선에서 공격을 주시해야 한다. 대부분의 이러한 공격들은 실제 개개의 접근통제 인스턴스 측면에서의 보안 설정을 위반하지 않으며, 이것은 보안 통제에서 행위적인 측면의 제어가 부족한 연유로 기인한다. 본 논문에서는 보안 통제에서 접근제어와 더불어 행위제어가 가능한 확장된 참조모니터를 제안한다.

1. 서론

보안운영체제(Trusted Operating Systems)는 커널에 보안 서브시스템을 구현하여 다양한 공격에 대비할 수 있는 향상된 운영체제를 의미한다. 보안 커널 방법은 TCB(Trusted Computing Base)개념의 참조 모니터(Reference Monitor)의 구현에 가장 적합한 방법이다 [1,2]. 최근의 보안운영체제 연구들은 대부분 기존 UNIX/LINUX운영체제의 임의적 접근제어(DAC, Discretionary Access Control) 보다 향상된 접근제어를 위해 강제적 접근통제(MAC, Mandatory Access Control)이나 역할기반 접근통제(RBAC, Role-Based Access Control)모델을 수정하여 보안커널을 구현하고 있다.

어떠한 향상된 접근제어를 보안 커널에 추가하더라도 대부분의 프로토타입들이 실시간에 프로세스의 권한을 전이시킬 수 있는 메커니즘을 채택하는 경향이 있다. 예를 들어 SELinux[3]의 domain transition이나 GRSecurity[4]의 role transition등이 그러한 메커니즘이며 이는 DAC 측면에서 Linux의 set-user-id 메커니즘과 유사하다. 이러한 메커니즘들은 공통적으로 어떤 특정한 목적을 위하여 임시적으로나 영구적으로 권한을 전이하는 기능을 수행한다. 예를 들어 관리자 소유의 패스워드 파일을 일반 사용자가 수정해야 하거나, 일반 사용자가 하드웨어 디바이스에 임시적으로 접근해야 하는 경우에 이러한 메커니즘이 필요하다. 하지만 우리는 이러한 메커니즘이 버퍼오버플로우 같은 프로그램 수행시간 공격에 의해 공격자가 다른 권한을 획득하는 용도로 사용될 수 있다는 것을 많이 보아왔다. 공격을 접근 인스턴스 단위로 분석을 해 보면, 각 접근 인스턴스는 주어진 접근통제 설정을 위반하지 않는다는 것을 발견할 수 있고, 단지 행위적인 의미를 변경시켜 공격을 수행하는 것을 볼 수 있다. 어떠한 형태의 접근통제가 설정되더라도 보안 통제에 행위적인 제어가 추가되지 않으면 이러한 공격의 위험성이 존재할 수 있다.

	Intended Behavior	Exploited Behavior
1	(john,john,john,*)	(john,john,john,*)
2	(john,john,john,execve(./prog))	(john,john,john,execve(./prog))
3	(john,root,root,*)	(john,root,root,*)
4	(john,root,root,read(buffer))	(john,root,root,read(buffer))
5	(john,root,root,*)	(john,root,root,*)
6	(john,root,root,exit(0))	(john,root,root,setreuid(0,0))
7	(john,john,john,*)	(root,root,root,*)
8	(john,john,john,*)	(root,root,root,execve(/bin/sh))

표 1. Stack Oveflow 공격의 접근 인스턴스 분석

	System Program	Attacker's Program
1	(root,root,root,access(/tmp/X))	
2		(john,john,john,unlink(/tmp/X))
3		(john,john,john, symlink(/etc/passwd, /tmp/X))
4	(root,root,root,open(/tmp/X))	
5	(root,root,root,write(/tmp/X))	

표 2. TOCTTOU 공격

본 연구에서는 접근제어와 더불어 행위제어가 가능한 참조모니터를 제안하고, 보안 오토마타 [8] 기반의 행위제어를 Linux에서 구현하였다.

2. 프로그램 공격의 접근 인스턴스 분석

본 절에서는 단일 프로세스 측면에서 Stack 오버플로우 공격[5]과 프로세스간 동시성과 관계되는 TOCTTOU binding 공격[6]을 접근 인스턴스 측면에서 살펴본다.

일반적인 Stack 오버플로우 공격은 메모리 버퍼의 바운드를 체크하지 않는 버그를 내제하는 Setuid 프로그램을 실시간에 공격하여 컴파일 시간에 바이너리에 코딩된 오퍼레이션이 아닌, 공격자가 원하는 오퍼레이션을 수행한다. 이때 공격당한 프로그램은 Setuid가 설정되어 있기 때문에 수행시간 동안에는 프로그램 소유자의 권한으로 수행이 되며 소유자가 root인 경우에는 그 위험성이 크다. 표 1 에서는 예제프로그램이 일반적인 수행과 공격당한 경우의 수행에 대한 단위 접근 컨텍스트를 (ruid, euid, suid, perm)로 나타내었다. 왼쪽은 프로그램의 정상적인 행위이고 오른쪽은 Stack Overflow 공격을 통하여 6행 이후의 행위가 컴파일 타임 때 정해진 오퍼레이션이 아닌 공격자의 코드로써 ruid를 root로 설정하고 쉘을 실행시킴으로써 일반유저(john)로부터 임시적으로 상승된 root권한을 유지하여 공격자 임의의 코드를 수행할 수 있다는 것을 보여준다. 중요한 점은 공격당한 경우 각각의 접근 컨텍스트가 모두 DAC관점의 접근제어설정을 벗어난 것이 아니라 stack overflow을 통한 행위변화를 통한 공격이라는 점이다.

TOCTTOU(Time-Of-Check-To-Time-To-Use) 공격은 어떠한 오브젝트의 레퍼런스를 체크하는 과정과 체크된 레퍼런스를 사용하는 사이에 아무런 일이 없을 것이라는 프로그래머의 잘못된 가정으로 인하여 발생한다. 하지만 운영체제 스케줄러의 영향으로 두 시간사이에 공격자의 행위가 수반될 가능성이 있다. 표 2 에서는 이러한 예를 보여준다. 왼쪽 열의 시스템 프로그램이 어떠한 파일의 레퍼런스를 체크한 후 쓰기를 하는 데, 공격자가 몇 번의 시도를 통해 access와 open사이에 해당 레퍼런스에 대한 unlink와 symlink를 수행하여, 결과적으로 자신의 권한으로는 불가능하나 시스템 프로그램의 권한(root)으로 쓸 수 있는 파일에 대하여 특정한 내용을 삽입할 수 있다. Stack Overflow의 경우와 마찬가지로, 표 2에 나타난 개개의 모든 접근 인스턴스는 일반적인 경우 가능한 인스턴스 집합으로 행위 적인 측면에서만 공격으로 간주된다.

3. 확장된 참조 모니터

본 절에서는 2절에서의 관찰을 토대로 기존의 접근통제를 주로 하는 참조모니터 개념을 확장하여 행위 제어까지 가능한 보안 제어를 할 수 있는 프레임웍을 제시한다.

일반적인 접근제어를 (user, operation, object)의 집합으로 구성하였으며 이것은 access matrix개념을 반영한다. DAC과 MAC의 경우 일반적으로 대응하여 사용하는 개념이며 최근 RBAC의 access matrix에 대한 해석[7]에 바탕으로 하여 일반적인 접근통제를 반영한다.

정의 1. 접근 제어 시스템

- $USER, OPR, OBJ$: the set of users, operations and objects
- $AR \subseteq USER \times OPR \times OBJ$: the set of access rights configured by access control policy.
- $access(u: USER, op: OPR, ob: OBJ) \rightarrow \{true, false\}$: On invocation of an operation *op* with an object *ob* by a user *u*, it returns *true* for a permitted access and, *false* for rejected accesses.

◆ $(\forall u: USER, \forall op: OPR, \forall ob: OBJ)\, access \Rightarrow [(u, op, ob) \in AR]$

확장된 참조 모니터에서 행위제어는 접근 행위의 연속에 (access trace)에 대한 속성을 정의하여 프로그램 수행시간에 검증하는 방법을 사용한다. 행위에

대한 명세는 간단한 Büchi Automata $A=(\Sigma, Q, q_0, \delta)$로 정의한다. Q는 셀 수 있는 오토마타 상태이며, $q_0 \in Q$는 초기상태이다. δ는 전이함수($\delta: Q \times \Sigma \rightarrow 2^Q$)이다. Σ는 보안운영체제의 접근객체에 대한 시스템 콜 오퍼레이션들의 집합이다. 정의된 명세에 벗어난 행위들에 대하여 행위 명세는 false를 리턴 한다.

정의 2. 행위 제어 명세 정의

- $STATE_i$: a finite set of state variables.
- $STATE_i$: a finite set of Boolean guarded commands of the form $guard \rightarrow command$.
- $BEHAV_i=(STATE_i, TRANS_i)$: a behavior and it is itself a function which returns the result of guarded command on its input symbol.
- $BEHAV=\{b \mid \exists i[b=BEHAV_i]\}$: the set of behavior in overall system.
- $behave(b: BEHAV, op: OPR, ob: OBJ) \rightarrow \{true, false\}$: it returns the result of a Boolean guarded command in the behavior specificating in b on a current input symbol (op, ob).

운영체제에서 행위는 프로세스에 의해 수반된다. 일반적으로 단일 프로세스가 여러 행위에 대한 제어를 받을 수 있지만, 병행적인 프로세스의 움직임을 제어하기 위해 하나의 행위명세가 여러 프로세스와 연관을 가질 수도 있다.

정의 3. 행위의 프로세스 할당

- $PROC$: the set of processes.
- $exec(u: USER) \rightarrow 2^{PROC}$: currently executing processes activated by a user u.
- $PBA \subseteq PROC \times BEHAV$: a many-to-many process-to-behavior assignment.

보안제어는 다음 함수로 정의된다.

정의 4. Security Decision

- $sdf(u: USER, p: PROC, op: OPR, ob: OBJ) \rightarrow \{true, false\}$: the security decision which returns a true for a permitted access instance and return false for rejected accesses.

◆ $(\forall u: USER, \forall p: PROC, \forall op: OPR, \forall ob: OBJ)$
$sdf \Rightarrow [p \in exec(u) \wedge (p, b) \in PBA \wedge access(u, op, b) \wedge behave(b, op, ob)]$

4. 행위 제어 및 구현

본 절에서는 확장된 참조 모니터에 적용되는 행위 명세에 관한 예제들을 제시한다. 실제적인 예를 위해 접근통제는 일반적인 DAC을 가정한다.

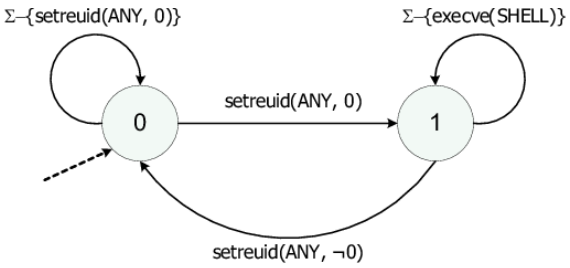

그림 1. Euid 0 후 쉘 수행 한정

4.1 권한 상승 후 쉘 수행에 대한 제한

Stack overflow와 같은 수행시간 공격들은 대부분 권한 상승 후, 쉘을 수행하여 상승된 권한을 유지한다. 예를 들어 *;setreuid(0,0);*;execve(SHELL)[1]과 같은 수행 시퀀스를 포함한다. 따라서 행위 참조 모니터는 setreuid 함수에 의해 euid 가 super user의 ruid와 같게 된다면 향후 쉘을 수행하는지 관찰을 해야 한다. 만약 상승된 권한으로 작업을 수행 후 정상적으로 권한이 원래의 권한으로 되돌아온다면 참조 모니터는 관찰 상태를 초기로 복원하나 쉘의 수행을 감지한다면 프로그램을 종료시켜야 한다[그림 1]. 이러한 수행 특성을 명세화하면 다음과 같다:

4.2 Chroot 회피에 대한 제어

For a process p, $(b, p) \in PBA$ where $b = (\{i : 0\},$
$\{op = \neg setreuid(ANY, 0) \wedge i = 0 \rightarrow$ return true,
$op = setreuid(ANY, 0) \wedge i = 0 \rightarrow i = 1$; return true,
$op = \neg execve(SHELL) \wedge i = 1 \rightarrow$ return true,
$op = setreuid(ANY, \neg 0) \wedge i = 1 \rightarrow i = 0$; return true,
otherwise return false $\})$

chroot 메커니즘은 보통 원래의 root 파일 시스템을 보호하기 위하여 다른 계층의 파일시스템 레이어를 제공하여 임의의 프로세스가 실제 root 파일 시스템에 대한 접근을 제어하기 위하여 사용한다. 하지만 이러한 보호 레이어는 다음과 같은 행위에 대하여 임의의 프로세스는 해당 보호 레이어를 탈출할 수 있다[9]: open(temp_dir);chroot(temp_dir); chdir("..")*;chroot(".");execve(SHELL).

그림 2는 이러한 취약점에 대한 안전한 수행에 대한 행위 명세를 표현한 것이다. 일단 choort에 의하여 non-root 디렉토리로 프로세스가 한정이 되면,

1) *(Asterisk)는 임의의 오퍼레이션 시퀀스를 의미한다. 만약 *가 open()*과 같이 오퍼레이션 바로 뒤에 붙게 되면, 그것은 해당 오퍼레이션을 여러 번 수행한다는 의미이다.

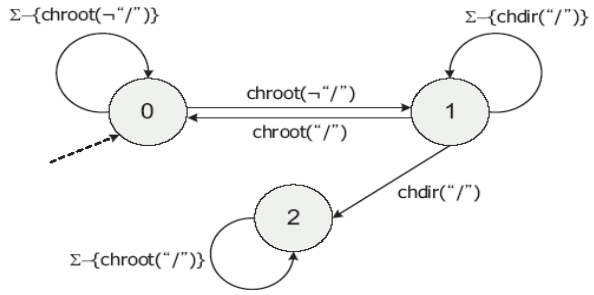

그림 2. chroot 탈출에 대한 제어

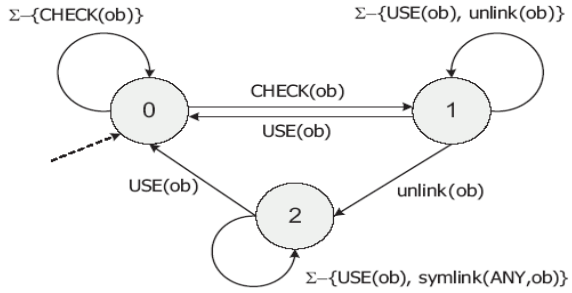

그림 3. TOCTTOU 제어

행위 참조 모니터는 root 디렉토리를 참조하는지 살펴보는 상태로 들어간다 (state 1). 그림의 State 2에서 chroot("/")를 하는 것은 행위 명세에 위배되는 것이다. 이 행위 명세를 표현하면 다음과 같다.

For a process p, $(b,p) \in PBA$ where $b = (\{i:0\},$
$\{op = \neg chroot(\neg"/") \wedge i = 0 \rightarrow$ return true,
$op = chroot(\neg"/") \wedge i = 0 \rightarrow i = 1$; return true,
$op = \neg chdir("/") \wedge i = 1 \rightarrow$ return true,
$op = chroot("/") \wedge i = 1 \rightarrow i = 0$; return true,
$op = chdir("/") \wedge i = 1 \rightarrow i = 2$; return true,
$op = \neg chroot("/") \wedge i = 2 \rightarrow$ return true,
otherwise return false $\})$

4.3 TOCTTOU 공격에 대한 제어

그림 3은 하나의 참조객체 ob에 대한 TOCTTOU 공격에 대한 행위 명세를 표현한다. 해당 명세는 임의의 두 프로세스간의 *;access(ob);*;unlink(ob);*; symlink(X,ob);*;open(ob);* 행위에 대한 제어이다. TOCTTOU 공격 자체가 하나 이상의 프로세스가 관여하기 때문에 PBA 관계는 하나의 행위 명세에 대하여 관계되는 프로세스 다수가 할당된다.

For two arbitary processes $p1$ and $p2$,
$(p1,b) \in PBA$ and $(p2,b) \in PBA$ where
$b = (\{i:0\},$
$\{op = \neg CHECK(ob) \wedge i = 0 \rightarrow$ return true,
$op = CHECK(ob) \wedge i = 0 \rightarrow i = 1$; return true,
$(op = \neg USE(ob) \wedge op = \neg unlink(ob)) \wedge i = 1 \rightarrow$ return true,
$op = USE(ob) \wedge i = 1 \rightarrow i = 0$; return true,
$op = unlink(ob) \wedge i = 1 \rightarrow i = 2$; return true,
$(op = \neg USE(ob) \wedge op = \neg symlink(ANY,ob)) \wedge i = 2 \rightarrow$ return true,
$op = USE(ob) \wedge i = 2 \rightarrow i = 0$; return true,
otherwise return false $\})$

4.4 시스템 구현

위의 명세들은 3절에서 제안한 확장 참조 모니터를 리눅스 커널 2.6.12.3에서 구현 및 실험하였다. 각 명세들에 대한 공격테스트에 대해 방어가 가능함을 확인하였다. HBench OS [10]를 통해 프로세스 생성에 관여하는 fork 및 execve의 오버헤드는 3.8% 였으며, 나머지는 각 시스템 콜마다 최소 0.1%에서 최대 4.3%의 오버헤드를 확인하였다.

5. 결론

본 논문에서는 보안운영체제를 위하여 기존의 참조모니터를 확장하여 접근 및 행위제어가 가능한 참조모니터를 제안 및 구현 하였다. 구현은 4절에서 제시한 공격 이외에 다른 여러 가지 명세를 보안관리자가 설정할 수 있다. 향후 연구로, 시제논리를 이용하여 접근 및 행위제어가 가능 수행시간 검증 체계를 참조모니터의 메커니즘으로 적용할 예정이다.

참고문헌

[1] Dept. of Defense (USA), Department of Defense Trusted Computer System Evaluation Criteria, Dept. of Defense Standard (DOD 5200.28-STD), Library Number S255, 711, 1985.

[2] E. G. Amoroso, Fundamentals of Computer Security Technology, AT&T Bell Laboratories, Prentice-Hall PTR, 1994.

[3] P. Loscocco, and S. Smalley, Integrating Flexible Support for Security Policies into the Linux Operating System, In Proc. of the FREENIX Track: 2001 USENIX Annual Technical Conference (FREENIX'01), 2001.

[4] B. Spengler, Increasing Performance and Granularity in Role-Based Access Control Systems (A Case Study in Grsecurity) (http://www.grsecurity.net/)

[5] Alphe One, Smashing The Stack For Fun And Profit, Phrack Magazine Vol.7 Issue. 49, File 14 of 16, 1996.

[6] M. Bishop and M. Dilger, Checking for Race Conditions in File accesses, Comput. Syst., Vol. 9, No. 2, pp. 131-152, 1996.

[7] G. Saunders, M. Hitchens, and V. Varadharajan, Role-Based Access Control and the Access Control Matrix, LNCS, Vol. 2836, pp. 145-157, 2003.

[8] F. B. Schneider, Enforceable Security Policies, ACM Trans. on Information and System Security, Vol. 3, No. 1, pp. 30-50, Feb. 2000.

[9] Simes, How to break out of a chroot() jail, 2002.
http://www.bpfh.net/computing/docs/chroot-break.html

[10] HBench-OS Operating System Benchmarks
http://www.eecs.harvard.edu/vino/perf/hbench