정보학연구 제4권 제3호 2001년 9월(pp.139-154)
The Journal of Information Technology, Vol.4, No.3

역할기반 접근제어에서의 사용자 수준의 위임기법에 대한 Rule-Based Framework

박종화*

요 약

현재의 역할기반 시스템에서는 보안관리자가 역할에 사용자를 지정한다. 이는 분산 환경에서 보안관리자의 계속적인 참여로 인해 관리의 어려움을 증가시키고 있다. 역할기반 위임 기술은 각 사용자의 위임을 갖는 분산 환경하에서 RBAC을 적용하게 하는 수단을 제공한다. 역할기반 위임의 기본 개념은 사용자가 자신의 역할의 권한을 다른 사용자에게 부여하여 그로 하여금 자신의 일을 수행하게 하는 것이다. 본 논문은 사용자가 새로운 위임 역할을 생성하여 자신의 역할 권한을 위임하는 사용자 수준의 역할기반 위임 모델에 대한 rule-based framework을 제시한다. 또한 보안 정책을 규정하고 적용하기 위한 rule-based 언어를 소개한다.

Ⅰ. 서론

역할기반 접근제어(RBAC : Role-Based Access Control)의 개념은 1970년대에 다중 사용자와 다중 응용을 위한 온라인 시스템으로부터 시작되어 최근 들어 기존의 임의적 접근통제(DAC : Discretionary Access Control) 및 강제적 접근통제(MAC : Mandatory Access Control)에 비하여 정교한 유연성을 제공함으로써 관심이 집중되고 있다[1]. RBAC의 기본 개념은 역할과 권한이 지정되고, 사용자는 적절한 역할에 지정된다는 것이다. 이것은 권한의 관리를 매우 쉽게 해준다. 역할은 조직내의 다양한 작업 함수에 대해 생성되고, 사용자는 그들의 자격과 책임에 기초하여 역할에 지정된다. 사용자는 하나의 역할에서 다른 역할로 쉽게 재 지정될 수 있고, 역할은 새로운 응용으로써 새로운 권한을 부여받을 수 있다. 또한 필요에 의해 권한이 역할로부터 회수될 수도 있다.

현재 RBAC 시스템에서 위와 같은 역할에 대한 사용자 지정이 보안관리자에 의해 이루어지고 있다. 보안관리자에 의한 사용자의 역할지정은 분산 환경 하에서 보안관리자의 계속적인 참여로 인해 관리에 대한 어려움을 증가시킬 수 있다. 최근 역할기반 위임(role-based delegation)에 대한 연구가 활발하게 진행되고 있는데, 이는 위임이 행정적인 일을 분산시킬 수 있는 수단을 제공하기 때문이다. 따라서 역할기반 위임은 어떤 기능을 수행할 수 있는 한 사용자의 역할에 대한 권한을 다른 사용자에게 위임하는 것으로 RBAC을 분산환경에 적용하기 위한 수단을 제공한다.

최근 RBAC에서 이러한 위임을 다루는 연구들이 수행되고 있는데, 위임에 관련된 연구는

* 세명대학교 소프트웨어학과

관리자가 역할에 부여한 권한을 조정함으로써 위임을 구현하거나[3,4,5], 위임이 용이한 정책을 구현하지만, 그로 인해 발생하는 보안의 문제를 다루지 않는다[1,6]. 또한 현재 위임을 다루는 연구들은 사용자 수준의 위임을 적절히 구현하고 있지 못하다.

본 논문은 현실세계에서 발생하는 위임의 의미를 유지하면서 RBAC 구조를 통해서 구현된 위임 모델[7]에 기초한 rule-based framework을 제시한다. 일반적으로 rule-based system은 명백한 rule들의 집합에 의해서 그 행위가 결정된다. 이 framework은 사용자 수준의 위임기법과 정책을 규정하기 위한 rule-based 언어를 포함한다.

이 논문의 나머지는 다음과 같이 구성된다. 2장에서는 이 논문의 동기와 관련된 연구들이 다루어지고, 3장에서는 위임과 철회를 포함하는 사용자 수준의 역할기반 위임 모델[7]에 대해 설명되어진다. 또 4장에서는 위임 정책과 철회 정책을 적용하고, 표현하는 rule-based specification 언어의 semantics에 대해 다루어진다. 마지막으로 5장에서 결론을 맺는다.

II. 동기와 관련된 연구

2.1 연구의 동기

위임에 대한 요구는 한 사용자가 자원에 접근함에 있어서 다른 사용자의 입장에서 행동할 필요가 있을 때 일어난다. 문헌상에 많은 정의가 있지만, 가장 일반적인 위임의 정의는 한 시스템 안에서 한 존재가 어떤 기능을 수행할 수 있는 자신의 권한을 다른 존재에게 위임하여, 다른 존재가 그 기능을 수행할 수 있도록 하는 것을 말한다. 위임에는 여러 가지 종류가 제시되어 왔는데, 가장 일반적인 위임의 종류로는 사용자 대 기계, 사용자 대 사용자, 그리고 기계 대 기계의 위임이 있다[2].

위임은 두 개의 견해로 정리되는데[8], 행정적 수준 위임과 사용자 수준의 위임이 그것이다. 행정적 수준의 위임은 모든 위임을 보안관리자가 담당하는 것을 말하고, 사용자 수준의 위임은 사용자가 자신의 조정 하에서 자원 접근에 대한 권한을 위임하는 것이다. 이 두 견해 모두에서 개개인의 사용자의 권한 남용을 막기 위해 사전에 정의된 위임 정책을 적용하는 것이 필요하다.

Barka와 Sandhu[9]은 위임과 관련해서 위임의 특성들을 구별하는데, 그 특성들은 영속성(permanence), 무결성(totality), 그리고 위임의 단계(levels of delegation) 등이다. 영속성은 위임 기간의 입장에서 위임의 종류를 의미하며, 무결성은 어떻게 완전하게 그 역할에 부여된 권한을 위임하느냐에 대한 것이고, 위임의 단계는 각 위임이 그 이상 더 또 얼마나 많이 위임될 수 있는 있는지에 대한 것이다.

이 논문에서는 사용자 대 사용자의 위임 즉 사용자가 다른 사용자에게 자신의 역할을 위임하는 사용자 수준의 위임에 초점을 맞춘다.

2.2 관련 연구

현재 RBAC 모델은 현실세계의 경험적 기업 모델에 맞추어 기업 조직에 대한 다양한 접근제어를 제공하기 위하여 많은 연구가 이루어지고 있다. 대부분의 연구는 이러한 RBAC 요소에 대한 세밀한 구성을 통하여 기업의 관리원칙에 적합하도록 RBAC을 구현하는 것이다.

최근 많은 개량된 RBAC 모델이 제안되었는데, 그 중 가장 일반적인 모델이 RBAC96[10]이다. RBAC96의 기본 개념은 권한이 역할에 연관되어지고, 사용자는 적절한 역할에 할당되어지는 것이다. 이때 사용자는 하나의 역할에서 다른 역할로 재 할당될 수도 있다. 또한 역할들에 새로운 권한이 부여될 수도 있다. 그리고 필요에 따라 권한은 역할로부터 쉽게 회수될 수도 있다.

Barka와 Sandhu[9]는 위임과 관련해서 RBDM0 모델를 제안하였다. 이 모델은 간단한 역할기반 위임 모델로서 회수, 계층적 역할에서의 위임, 부분 위임, 그리고 다단계 위임을 포함하고 있다. RBDM0 모델의 제한점은 이 모델이 위임의 각 구성 요소들 간의 관계에 대해서 겨냥하지 못하고 있는 것이다.

ARBAC97[11]는 보안 관리자에 의한 사용자 역할 할당에 대해 URA97를 제안하였다. 기본 개념은 RBAC 자체를 관리하기 위해 RBAC을 사용할 수 있다는 것이다. 이것은 많은 행정 적인 편리성과 확장성을 제공하고 있지만, 보안 관리자의 계속적인 참여로 분산 시스템에서는 관리에 대한 어려움을 증가시킬 수 있다.

본 논문은 현실 세계를 반영하는 위임의 의미 즉 사용자 수준의 위임 RBAC 구조를 통해 구현한 위임 모델[7]에 기초한 rule-based framework을 제시한다.

Ⅲ. 사용자 수준의 위임 모델과 형식적 표현[7]

제시한 모델에서 위임역할은 사용자가 위임할 권한의 집합으로 구성된 새로운 역할을 생성한 것이다. 위임역할은 생성한 사용자가 사용자 지정권한을 가지도록 하여 다른 사용자에게 권한을 가지도록 하여 다른 사용자에게 권한을 위임할 수 있도록 한다. 또한 권한 위임 시에 역할의 일부분만을 위임할 수 있도록 역할을 여러 개의 작업으로 세분화한다. 따라서, 실제 역할-권한 지정은 작업-권한 지정으로 수정된다. 이러한 세분화된 작업은 실제로 일을 수행하는데 의미 있는 하나의 단위이어야 한다.

이렇게 위임의 수행을 보안 관리자가 직접 관리하지 않음으로서 사용자의 필요에 의해 언제나 간단히 이루어질 수 있다. 그러나 사용자 수준의 위임은 보안관리자에 의해 관리될 때보다 많은 보안의 위협을 가진다. 따라서 적절한 위임이 이루어지도록 시스템 상에서 제한을 두어야 한다. 위임받는 사용자에 대한 제한을 위해서, 각 사용자에 대하여 각각 범위 속성을 부여하고, 해당하는 범위에 따라 역할에 지정되도록 함으로써 위임된 역할 또한 해당 범위의 사용자에게만 위임되도록 하여 다른 도메인으로 정보가 유출되는 것을 방지한다. 다음절에서는 이 위임 모델을 구성하는 각 부분에 대하여 설명한다.

3.1 사용자 수준의 위임 모델

3.1.1 사용자 수준에서의 위임

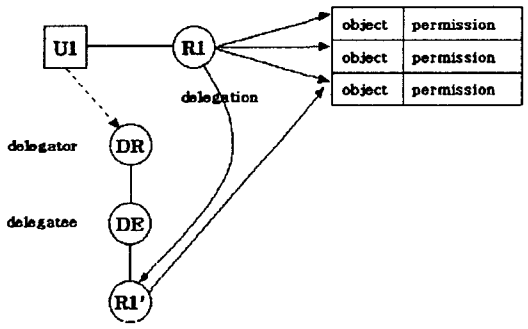
사용자로부터 사용자에게로의 권한 위임을 가장 잘 반영하고 있는 것은 임의적 접근제어(DAC)이다. 그러나 RBAC에서는 객체에 대한 소유권이 개개인의 사용자에게 부여되는 것이 아니기 때문에 그러한 DAC 정책을 그대로 적용시키기 어렵다. 또한 RBAC은 정책에 중립적인 성질을 갖는다. 즉 RBAC 구조를 사용하여 강제적 접근제어(MAC), 임의적 접근제어와 같은 접

근제어 정책을 구현할 수 있다. 실제로 [1]에서는 RBAC의 역할을 사용하여 DAC정책을 구현하였다.

사용자 수준의 위임을 RBAC상에서 적용하기 위해서 [1]의 정책을 부분적으로 적용한다. 즉 위임이 발생할 경우에 위임하는 역할에 대해서만 [1]에서 제시한 DAC 정책을 적용하도록 한다. DAC의 위임 모델은 권한-회수(Grant-Revoke) 모델이다. 다양한 DAC의 위임 정책이 존재하지만, RBAC으로 구현된 기업의 조직상에 이루어지는 위임에 대하여 가능한 모델을 가정하여 몇 가지로 제한한다.

우선 권한 위임시 발생할 수 있는 두 가지 모델은 다음과 같다.

- 단순 위임(simple delegation) 정책 : 위임자는 임의적 권한을 다른 객체에 부여할 수 있는 유일한 사람이다. 즉 해당 권한을 위임받은 사람이 다른 사람에게 또 다시 권한을 부여할 수 없다.
- 다단계 위임(liberal delegation) 정책 : 위임받은 사람은 또 다른 사람에게 위임받은 권한의 일부를 또 다시 위임할 수 있다.



(그림 1) 사용자 수준의 위임을 위한 RBAC 모델

다음으로 권한 회수와 관련된 모델은 다음과 같다.

- 권한 위임과 무관한 권한 회수(delegation independent revocation) : 권한을 부여한 사람과 관계없이 회수 권한을 가진 사람은 누구나 부여된 권한을 회수할 수 있다.

권한을 위임하려는 사용자는 위임할 권한을 포함하는 새로운 역할을 생성할 수 있다. 새로운 역할은 사용자가 이미 지정되어 있는 역할의 부분집합이 된다. 이렇게 생성된 역할은 역할을 생성한 사용자의 소유가 된다. [그림 1]은 사용자 수준의 위임을 위한 RBAC 모델을 보여준다. [그림 1]에서 사용자 U1이 자신에게 지정된 역할 R1의 일부분을 위임하려고 한다면, R1의 부분집합으로 구성된 역할 R1′을 생성한다. 이때 이 R1′의 소유자는 U1이 되고 R1′는 R1이 가진 권한의 일부를 수행할 수 있다. 이렇게 역할 R1′이 생성되면 자동적으로 R1′에 대한 DR, DE 역할이 생성된다. DR, DE, R1′의 역할 계층 관계는 [그림 1]에서 볼 수 있듯이 DR > DE > R1′의 관계를 갖는다.

각각의 역할이 갖는 의미와 권한, 그리고 위임역할과의 관계는 다음과 같다.

- DR(delegator) : 위임역할을 생성한 사용자에게 지정되는 역할이다. 최초 U1이 R1′을 생성하면 자동적으로 U1은 DR에 지정된다. DR은 다음의 권한을 갖는다.
  · Destroy_role() : 위임역할을 삭제한다. 즉, 위임이 더 이상 이루어지지 않도록 한다.
- DE(delegatee) : R1′ 역할을 위임하는 사용자에게 지정되는 역할이다. 이 역할에 지정된 사용자는 다른 사용자에게 권한을 위임할 수 있다. DE는 다음의 권한을 갖는다.
  · AssignUser_DE() : 다른 사람에게 위임

할 수 있는 권한과 함께 R1'의 권한을 위임할 사용자를 지정한다. 이 사용자는 DE 역할에 지정된다.

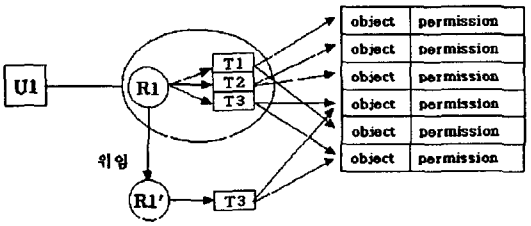
- RevokeUser_DE() : 부여된 사용자의 권한을 회수한다.
- AssignUser_role() : 실제 위임역할의 사용자를 지정한다.
- RevokeUser_role() : 위임역할에 지정된 사용자에 대한 지정을 해제한다.

- R1' : R1의 부분집합으로 이루어진 역할이다. 위임되어질 권한을 포함한다.

DR에 지정된 사용자 U1이 R1'에 사용자를 지정시킴으로써, R1'의 역할을 지정 받은 새로운 사용자가 실질적으로 U1으로부터 R1'의 역할을 위임받은 것처럼 R1'의 권한을 수행할 수 있게 된다.

3.1.2 지정된 역할의 부분집합

지금까지 역할의 일부를 위임할 수 있다는 것을 가정하였다. 그러나 RBAC 모델 상에서 사용자는 권한과 직접적으로 지정되어 있지 않고, 역할과 지정되어 있기 때문에 역할의 부분집합을 만들어낼 수 있는 방법이 없다. 실제 기업에서는 일부의 권한만을 위임하는 경우가 대부분이기 때문에 역할전체를 위임하는 것은 문제가 있다. 따라서 제안된 모델에서는 RBAC 모델이 갖는 역할의 의미를 약간 다르게 해석한다. 즉 역할에 할당된 권한을 직접 지정하는 것이 아니라, 역할에 할당된 일을 작업 단위로 세분화해서 권한을 지정시키는 것이다. 그렇게 되면, 작업 단위의 의무 분리 정책 적용이 가능해져서 업무의 효율을 높이는 효과를 얻을 수 있다. 또한 역할을 작업의 집합으로 구성함으로써 사용자가 위임할 수 있는 역할의 부분집합을 작업 집합의 일부분으로 지정할 수 있으므로 권한의 일부분을 위임할 수 있다.



(그림 2) 작업의 세분화된 단위로서의 역할과 부분집합의 위임

예를 들면, [그림 2]에서 프로그램 개발부의 R1역할이 프로젝트 리더라고 가정하자. 그러면 R1에 부여된 작업은 여러 가지가 될 수 있다. 우선 하위 역할의 작업을 감독하는 작업(T1)과 특정 모듈과 관련된 분석 및 설계(T2), 코딩(T3)등의 작업등이 부여되어 있다고 하자. 이 때, R1의 역할이 세부 작업의 집합으로 구성되어 있지 않다고 한다면, U1은 특정 모듈에 대한 코딩 작업만을 위임할 수 있는 방법이 없다. 그러나 [그림 2]와 같이 세분화된 작업의 집합으로 구성되어 있다면, 사용자는 위임하려고 하는 작업을 선택하여 위임역할 R1'을 생성할 수 있다. 따라서 T3 작업에 해당하는 권한만을 위임할 수 있다.

3.1.3 사용자의 범위 속성

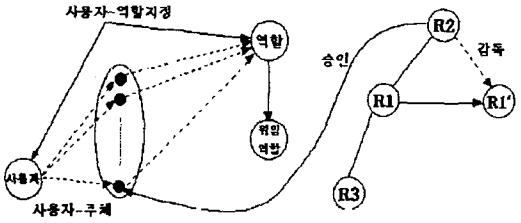
사용자가 생성한 위임역할을 통하여 권한이 위임되는 모델에서의 문제점은 위임의 전파(propagation)가 허가되지 않은 사용자에게 정보를 유출할 수 있다는 것이다. 따라서 정보의 유출을 막기 위해서 위임할 수 있는 대상에 대한 제한이 필요하다. 또한 이러한 대상을 제한하기 위한 기준은 다음과 같다.

- 의무분리 정책 : 의무분리 관계에 놓여있는 역할을 위임할 경우 위임하는 대상이 이미 의무분리 관계에 놓여 있는 다른 역할과 직접 또는 역할계층을 통해 간접적으로 지정되어 있다면 위임의 대상으로 적합하지 않다.
- 역할에 지정될 수 있는 사용자 수 : RBAC 허가 정책에 부여된 제약의 하나로 조직내의 어떤 역할은 일정 기간 동안에 일정 수의 멤버를 가져야만 한다. 예를 들면 부서의 장을 나타내는 역할이다. 많은 개인들이 이 역할에 지정될 수는 있지만, 주어진 한 시점에서 부서의 장은 단 한 명이어야 한다. 위임역할도 기존의 역할과 마찬가지로 이러한 속성을 유지하여야 한다.
- 도메인 내의 사용자 : 역할의 위임은 그 역할을 소유하고 있는 사용자와 관련된 도메인내에서 이루어지는 것이 보통이다. 예를 들면, 프로젝트1팀의 한 사용자가 자신의 권한을 프로젝트2팀의 누군가에게 위임해서는 안 된다. 프로젝트의 성격이 중요한 보안을 요구하는 경우에는 더더욱 그러하다. 같은 도메인, 같은 역할 계층상에서의 위임이 일어나는 것이 바람직하다.

3.1.4 범위 내에서의 위임 감독

위임될 대상에 사용자의 범위 속성과 기존 역할이 갖는 제약조건을 상속하여 제한함으로써, 보안 관리자의 직접적인 지정 없이 시스템 상에서 위임 대상에 대한 제한적인 선택을 가능하게 한다. 그러나 기준을 만족하는 대상간의 위임에도 문제가 있을 수 있다. 또한 위임 받은 사용자가 그들의 의무를 적절하게 수행하지 않을 위험이 존재한다. 따라서 사용자의 행위가 특히 역할계층상의 상위자와 같은 또 다른 사용자에 의해 감독, 검토되어야 할 필요가 있다. 같은 범위, 즉 같은 역할 계층상에서 일어나는 위임의 경우에도 이러한 위임이 적합한 것인지에 대한 판단이 필요하다.

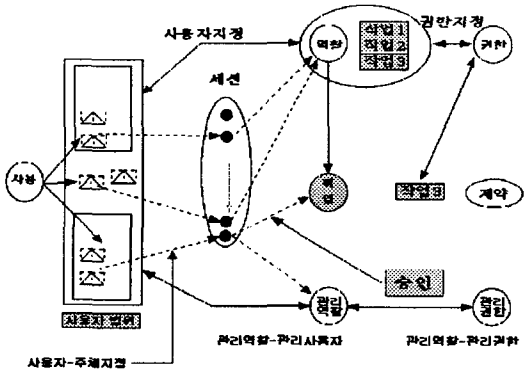
이러한 위임의 적합성을 판단하기 위하여, 상위 역할 또는 보안관리자가 사용자 범위 내에서 이루어지는 위임에 대한 적합성 여부를 승인하도록 한다. 위임 역할은 새로 생성된 역할이기 때문에 다른 역할과의 계층관계를 갖지 않는다. 따라서 그러한 역할을 감독할 만한 역할이 존재하지 않는다. 이러한 점을 보완하기 위해서 위임역할의 모태가 되는 기존의 역할이 갖는 역할 계층에서 상위 역할이 가지는 감독 권한에 대한 정보를 상속받는다. 따라서 새로 생성된 위임역할에 대한 감독은 기존 역할에 대한 감독을 하는 역할 계층상의 상위 역할이 위임역할에 대한 감독을 하게 된다. [그림 3]



(그림 3) 위임역할에 대한 감독 역할 및 위임역할의 활성화

3.2 위임 모델의 형식적 표현

사용자 수준의 위임 모델은 [그림 4]와 같이 정의된다. 다음은 이 위임 모델에 대한 형식적 표현이다.



(그림 4) 사용자 수준의 위임모델

3.2.1 기본 집합 및 함수

이 모델에서 사용할 집합은 다음과 같다.

U : 사용자 집합
R : 역할 집합
T : 작업 집합
P : 권한 집합
AR : 관리역할 집합
AP : 관리 권한 집합
S : 주체 집합

<정의 1>

- UA(r)은 역할 r에 지정된 사용자의 집합을 나타낸다.

$\forall u \in U, \forall r \in R, UA(r) : R \rightarrow 2U$

- RU(u)는 주어진 사용자에 대해 허가된 역할의 집합을 나타낸다.

$\forall u \in U, \forall r \in R, RU(u) : U \rightarrow 2R$

- Inherits는 역할간의 상속관계를 나타낸다.

$(r1, r2) \in Inherits \Rightarrow r1 \rightarrow r2$또는 $r1 \leq r2$ (r2가 상위역할)

- TP(t)는 작업 t에 지정된 권한의 집합을 나타낸다.

$\forall p \in P, \forall t \in T, TP(t) : T \rightarrow 2P$

- RT(r)은 역할 r에 지정된 세부 작업의 집합을 나타낸다.

$\forall r \in R, \forall t \in T, r=\{t|t \in RT(r)\}$
$RT(r) = \{t1, t2, ...,tn\}$
$\Rightarrow r = \{t1, t2, ..., tn\}$ (n : 0.n 인 정수)

- cardinality(r)은 역할r의 사용자 지정수를 나타낸다.

$cardinality : R \rightarrow N$

- SU(s)는 사용자와 주체의 지정, 즉 주체 s와 관련된 사용자를 나타낸다.

$\forall s \in S, SU(s) \rightarrow u$

- AR(s)는 주체 s가 활성화한 역할의 집합을 나타낸다.

$\forall r \in R, \forall s \in S, AR(s) \rightarrow 2R$

<정리 1> 주체는 관련된 사용자에게 허가된 역할만을 활성화할 수 있다.

$\forall s \in S, \forall u \in U, \forall r \in R, \forall r \in AR(s) \Rightarrow SU(s) \in UA(r)$

3.2.2 위임역할의 생성

위임할 역할을 생성하기 위해 사용자는 자신이 가진 역할의 일부분을 위임 역할로 생성할 수 있는 권한을 갖는다.

<정의 2> 역할 r에 지정된 사용자 u는 다음의 권한을 갖는다.

∀r1∈R, ∀u∈UA(r) :
Create_DRole(r1, r2) s.t. r1∈RU(u)

r2는 생성된 위임 역할의 이름이다. 앞으로 역할 r1에서 생성된 위임역할에 대해서는 DRole(r1)이라고 표기한다. 이 권한을 가지고, 사용자 u는 자신이 가진 역할중의 일부분을 위임을 위해 새로운 역할로 생성할 수 있다. 역할의 일부분은 각 역할을 이루는 세부 단위작업을 선택하여 이루어진다.

<정의 3> 사용자 u는 자신이 가진 역할 r의 일부분을 위임역할로 지정할 수 있다.

∀u∈UA(r) :
Select_Subjob(DRole(r), t) s.t. t∈subset of RT(r)

그러면 위임 역할을 관리하기 위하여 시스템에서 자동적으로 다음의 역할과 역할에 해당하는 권한을 생성한다.

Create_DR(DRole(r)) : 위임역할의 위임자역할 DR을 생성한다.
Create_DE(DRole(r)) : 위임역할의 위임받는 사용자역할 DE를 생성한다.

여기서 DRole은 Create_DRole()을 통해 생성된 위임역할의 이름이다. 각각의 역할은 다음과 같은 권한을 갖는다. 함수와의 혼동을 피하기 위해서 DR, DE, DRole로 간단하게 표시한다.

<DR>
Destroy_DRole(DRole)

이렇게 자동 생성된 역할 DR은 위임 역할을 생성한 사용자에게 자동적으로 지정되어 역할을 위임하거나 회수할 수 있는 권한을 갖도록 한다.

<정리 2> 생성된 DR 역할은 위임역할을 생성한 사용자 u에게 지정된다.

∀r∈R, cardinality(DR) = 1, UA(DR) = u

<DE>
AssignUser_role(u, DRole) : DRole에 사용자를 지정한다.
RevokeUser_role(u, DRole) : DRole에서 사용자를 제거한다.
AssignUser_DE(u, DRole) : DE 역할에 사용자를 지정한다.
RevokeUser_DE(u, DRole) : DE 역할에서 사용자를 제거한다.

각각의 역할 DRole과 DE, DR 역할은 계층관계와 권한상속을 통하여 역할위임과 위임의 정도를 관리한다.

<정리 3> 위임과 관련된 DRole, DE, DR 역할의 계층 관계는 다음과 같다.

DRole → DE → DR (DRole ≤ DE ≤ DR)

<정리 4> 각각은 역할 계층에 의해 권한을 상속받는다. 따라서 DR, DE 역할에 지정된 사용자는 DRole의 권한을 수행할 수 있다.

DR, DE, DRole, $\forall u \in U$: $(u \in UA(DE) \Rightarrow u \in UA(DRole)) \wedge (u \in UA(DR) \Rightarrow u \in UA(DE))$

<정리 5> DE 역할은 기본적으로 DRole 역할의 사용자 지정수를 상속받는다.

cardinality(DE) = cardinality(DRole)

<정리 6> DE 역할의 사용자 지정수(cardinality)를 조정하여 위임받는 사용자의 수를 제한한다.

$\forall r \in R$, $|UA(DE)| \leq$ cardinality(DE)

<정리 7> 이 때, DE 역할의 사용자 지정수는 DRole 역할의 사용자 지정수를 초과할 수 없다.

$|\text{cardinality(DE)}| \leq |\text{cardinality(DRole)}|$

여기서, 위임에 대한 기본적인 가정은 최초 권한을 위임한 사람에 의해서 모든 위임된 권한이 회수되고(정리 2), 위임한 후에도 권한을 계속 수행할 수 있다는 것을 가정한다. 권한의 회수는 권한 부여와 무관한 회수(Grant-Independent Revocation)를 기본으로 한다.

3.2.3 사용자 범위

위임 역할을 생성하여 위임할 경우에 무결성과 비밀성과 같은 보안의 문제를 해결하기 위해서, 위임대상자에 대한 제한이 필요하고, 그러한 제한을 위해서 다음과 같은 범위 속성을 도입한다.

<정의 4> SA : 범위 속성 집합

<정의 5> 사용자와 역할은 범위 속성을 갖고, 각각은 같은 도메인을 갖는다.

$\forall$User_scope_attribute, Role_scope_attribute $\in SA$, $\forall u \in U$, $\forall r \in R$

°User_scope_attribute=user_scope(u)
사용자 범위속성의 집합을 나타낸다.
°Role_scope_attribute=role_scope(r)
역할 범위 속성의 집합을 나타낸다.

<정리 8> 사용자를 역할에 지정할 경우에 다음을 만족할 경우에만 지정될 수 있다.

$\forall u \in U$, $\forall r \in R$: $u \in UA(r) \Rightarrow$ user_scope(u) $\supseteq$ role_scope(r)

이러한 범위 속성은 위임역할에도 마찬가지로 적용된다. 즉 위임역할도 이러한 범위 속성을 갖는다.

<정리 9> DRole의 역할범위 속성은 위임역할을 만드는 근원 역할의 범위 속성을 상속받는다.

$\forall r \in R$, role_scope(DRole)=role_scope(r)

<정리 10> DRole의 경우에도 <정리8>을 만족해야 한다.

$\forall u \in U$, $\forall r \in$ DRole : $u \in UA(\text{DRole}) \Rightarrow$ user_scope(u) $\supseteq$ role_scope(DRole)

<정리 11> 각 DRole의 DR, DE역할은 해당하는 DRole의 역할범위를 상속받는다.

∀r∈DRole, ∀rsa∈role_scope_attribute :
(role_scope(DRole)=rsa ⇒
role_scope(DE)=rsa) ∧
(role_scope(DE)=rsa ⇒ role_scope(DR)=rsa)

다음은 역할위임과 의무분리 속성에 대한 해석이다.

<정의 6> SSD : R×R, 역할간에 정적 의무분리 관계를 나타낸다.

<정의 7> DSD : R×R, 역할간의 동적 의무분리 관계를 나타낸다.

DRole(i)은 역할 i로부터 생성된 위임역할을 의미한다. 다음은 각각 정적 의무분리와 동적 의무분리 관계를 나타낸다.

<정리 12> DRole(i)에 대한 정적 의무분리 관계

∀u∈U, ∀i, j∈R, ∀r∈DRole(i) : u∈UA(DRole(i)) ∧ u∈UA(j) ⇒ (DRole(i), j) ∉ SSD

<정리 13> DRole(i)에 대한 동적 의무분리 관계

∀s, t∈S, ∀i, j∈R, ∀r∈DRole(i), s ≠ t :
DRole(i)∈AR(s) ∧ j∈AR(t) ∧ (DRole(i), j)∈DSD ⇒ SU(s) ≠ SU(t)

3.2.4 위임의 감독

주체는 관련된 사용자에게 허가된 역할만 활성화할 수 있다.

∀s∈S, ∀u∈U, ∀r∈R, ∀r∈AR(s) ⇒ SU(s)∈UA(r)

여기에 상위 역할에 의한 감독이 추가된다.

<정리 14> DRole(i)가 생성되면, 역할 I의 상위 역할 j는 다음의 권한을 갖는다.

∀i, j∈R, i≤j, ∃u∈UA(j), u has permit_delegation(DRole(i))

<정리 15> 위임역할에 지정된 사용자가 해당되는 위임역할을 활성화하기 위해서는 다음과 같은 조건을 만족해야 한다.

∀s∈S, ∀r∈R : DRole(r)∈AR(s) ⇒ SU(s)∈UA(DRole(r)) ∧ permit_delegation(DRole(r))

Ⅳ. RULE_BASED FRAMEWORK

앞장에서 이 모델은 일반 사용자가 그들의 역할을 위임하는 것을 허용하는 정책을 정의하였다. 또한 어떤 위임된 역할이 회수될 것인지에 대한 정책도 규정하였다. 이 장에서는 이와 같은 정책들을 적용하기 위해 사용되는 rule-based 언어가 설명된다. 이 rule-based 언어를 사용하는데는 2가지 이유가 있다. 첫째는 위임과 회수 관계가 선언적 규칙(declarative rule)으로 자연스럽게 표현될 수 있다는 것이고, 또 하나는 각 조직이 그 이상의 위임과 회수를 조정하기 위해 지역 정책(local policy)을 추가할 필

요가 있을 수 있다는 것이다. 선언적 규칙에 기초한 시스템에서는 그와 같은 지역 정책을 쉽게 포함하는 것을 허용한다.

4.1 언어

이 rule-based specification 언어의 주목적은 위임과 회수에 대한 권한을 적용하는 것이다. rule-based 언어는 rule과 logic을 묶는 선언적 언어[12]이다. 이 언어의 이점은 전체적으로 선언적이기 때문에 보안관리자가 정책을 정의하기에 쉽다는 것이다[2]. 이 언어는 절 논리(clausal logic)를 갖는 rule-based 언어이다.

<정의> 하나의 절(clause 또는 rule)은 다음과 같은 형태를 갖는다.

$$H \leftarrow F1\&F2\& \ldots \&Fn$$

여기서 H, F1, F2, ... , Fn은 boolean 함수들이다.

이 rule은 다음과 같이 읽을 수 있다.

to deduce H,
    deduce F1 and
    deduce F2 and
        ...
    deduce Fn

이 언어는 기본적으로 함수의 집합이다. 그 함수에는 이름과 변수의 집합과 반환 값(return value)을 갖는 함수, 다른 함수를 변수로 갖는 함수, truth-value를 반환하는 boolean 함수 등이 있다. 이 함수들은 3 부류로 분류되는데, 모델로부터 사상되는 함수(mapping system functions), utility 함수, 그리고 authorization 함수이다.

4.2 함수

함수들은 3 부류로 분류되는데, 언어의 함수로 사상된 모델의 system 함수 (즉 mapping system functions)가 [표1]에서 보인다. Utility 함수는 범용 boolean 함수로서, in, active, inheritUnum, inheritAttr 등이 있으며, [표2]에 정리되어 있다. authorization 함수는 권한 정책과 이 정책의 적용에 대해 정의하고 있다. authorization 함수는 기본적인 authorization 함수와 유도된 authorization 함수로 나누어진다. 사상 함수와 utility 함수의 의미는 각각 표에서 정의되어진다.

〈표 1〉 Mapping system 함수들

Mapping functions	system functions	semantics
UA(r)	UA : R → 2U	역할 r 에 지정된 모든 사용자의 집합을 return.
RU(u)	RU : U → 2R	주어진 사용자에 대해 허가된 역할의 집합을 return.
TP(t)	TP : T → 2P	작업 t 에 지정된 권한의 집합을 return.
RT(r)	RT : R → {t1, t2, ... , tn}	역할 r 에 지정된 세부 작업의 집합을 return.
cardinality(r)	cardinality : R→N	역할 r 에 사용자 지정수를 return.
SU(s)	SU : S → U	주체 s 와 관련된 사용자를 return.
AR(s)	AR : S → 2R	주체 s 가 활성화한 역할의 집합을 return.
inherits(r1, r2)	≤	역할간의 상속관계 즉 r1 → r2 또는 r1 ≤ r2 (r2가 상위역할)

〈표 2〉 Utility 함수들

Functions	Return value	semantics
in(a, b)	Truth value	b 가 a 의 부분 집합
active(u, r, s)	Truth value	주체는 관련된 사용자에게 허가된 역할만 활성화할 수 있다. 즉 사용자 u는 역할 r에 지정되었으며, 주체 s와 관련되었을 때 return. (SU(s) ∈ UA(r)).
inheritUnum(r1, r2)	Truth value	역할 r2는 r1의 사용자 지정수를 상속받는다. 즉 cardinality(r2) = cardinality(r1).
inheritAttr(r1, r2)	Truth value	역할 r2의 범위속성은 역할 r1의 범위속성을 상속받는다. 즉 role scope(r2) = role scope(r1)
static_seperable (DRole(r1)), r2)	Truth value	u∈UA(DRole(r1)) ∧ u∈UA(r2) ⇒ (DRole(r1), r2)∉SSD
dynamic_seperable (s1, s2)	Truth value	SU(s1) ≠ SU(s2)
permit_delegatable (u, DRole(r1))	Truth value	r1 ≤ r, ∃u∈UA(r) 즉 DRole(r1)가 생성되면, 역할 r1의 상위역할 r은 위임의 권한을 갖는다

4.3 기본적인 authorization rule

기본적인 authorization rule은 H← 의 형태를 갖는다. 이 규칙들은 이 모델에서 이미 정의된 보안 정책들이다.

Rule 1. 위임 역할을 생성하는 rule :

Creat_DRole(r1, r2) ←

여기서 r2는 역할 r1에서 생성된 위임 역할의 이름으로, DRole(r1)과 같다.

Rule 2. 역할의 일부분을 위임 역할에 지정하는 rule :

Select_Subjob(DRole(r), t) ←

여기서 t∈RT(r)의 부분집합

Rule 3. 위임역할의 위임자역할 DR을 생성하는 rule :

Creat_DR(DRole(r)) ←

Rule 4. 위임역할의 위임받는 사용자역할 DE를 생성하는 rule :

Creat_DE(DRole(r)) ←

Rule 5. 역할을 생성한 사용자에 지정되어 위임 역할을 회수하는 rule :

Destroy_DRole(DRole) ←

Rule 6. DRole에 사용자를 지정하는 rule :

AssignUser_role(u, DRole) ←

Rule 7. DRole에서 사용자를 제거하는 rule :

RevokeUser_role(u, DRole) ←

Rule 8. DE 역할에 사용자를 지정하는 rule :

AssignUser_DE(u, DRole) ←

Rule 9. DE 역할에 사용자를 제거하는 rule :

RevokeUser_DE(u, DRole) ←

4.4 정책을 적용하기 위한 유도된 Authorization rules

기본적인 authorization rules 은 모델에서 정의된 정책들을 규정한다. 그러나 사용자는 이 기본적인 authorization rules 을 통해서 위임이나 또는 회수에 대한 권한이 부여될 수 없다.

그 이유는 기본적인 rules 이 사용자보다는 역할에 기반으로 규정되어 있기 때문이다. 따라서 이 정책을 적용하기 위해서는 그 이상의 유도가 필요하다.

4.4.1 접근제어 정책의 적용

Rule 10. 접근 제어 rule :

```
allow(u, r, s) ← active(u, r, s) &
                 in(role_scope(r),
                 user_scope(u)) &
                 TP(RT(r))
```

여기서 u, r, s 는 사용자, 역할, 주체를 각각 나타낸다.

이 rule은 사용자 u는 주체 s 와 관련되었고, 사용자 범위 속성이 역할의 범위 속성에 포함되어 역할 r 에 지정되었음을 나타낸다. 또 이 역할은 세분화된 작업의 집합으로 구성되며, 작업과 관련된 권한은 작업 단위로 부여된다.

4.4.2 위임 정책의 적용

Rule 11. 유도된 사용자 수준의 위임 권한 rule :

```
der_delegate(u, u1, u2, r, r1, r2, s1,
             s2, t) ←
        active(u1, r1, s1)&
        active(u2, r2, s2)&
        in(role_scope(r1),
          user_scope(u1))&
        in(role_scope(r2),
          user_scope(u2))&
        Create_DRole(r1,
          DRole(r1))&

Select_Subjob(DRole(r1), t)&
  Create_DR(DRole(r1))&
        AssignUser_role(u2, DRole(r1))&
        AssignUser_DE(u2, DRole(r1))&
        inheritUnum(r1, DRole(r1))&
        inheritAttr(r1, DRole(r1))&

static_seperable(DRole(r1), r2)&
        dynamic_seperable(s1, s2)&
        permit_delegatable(u, DRole(r1))
```

4.4.3 위임 회수 정책의 적용

Rule 12. 위임 역할을 회수하기 위한 권한 rule

```
der_revoke_user_role(r1, r2, u2, s2) ←
        active(u2, r2, s2)&
        in(role_scope(r2),
         user_scope(u2))&

Destroy_DRole(DRole(r1))&
        RevokeUser_role(u2, DRole(r1)
```

Rule 13. 위임 역할을 위임받은 사용자 DE를 회수하기 위한 권한 rule

```
der_revoke_user_DE(r1, r2, u2, s2) ←
        active(u2, r2, s2)&
        in(role_scope(r2),
          user_scope(u2))&

Destroy_DRole(DRole(r1))&
        RevokeUser_DE(u2, DRole(r1))
```

V. 결론

본 논문에서 사용자 수준의 위임기법과

rule-based specification 언어에 대한 rule-based framework이 제안되었다. 사용자 수준의 위임 기법은 현실 세계에서 위임이 사용자에서 사용자에게로 일어난다는 것에 중점을 두고, 위임역할을 생성함으로써 RBAC 상에서 이러한 사용자 수준의 위임을 구현할 수 있는 방법을 제시한다. 또한 역할이 가진 속성 및 사용자 속성을 비교하고, 역할이 가진 제약을 적용함으로써, 위임역할을 지정할 수 있는 대상이 되는 사용자를 제한하여 비밀성과 무결성을 유지한다.

분산 환경 하에서의 보안 관리자에 의한 역할에의 사용자 지정은 보안관리자의 계속적인 참여로 관리의 어려움을 증가시킬 수 있다. 사용자 수준의 위임은 이런 의미에서 RBAC을 분산 환경에 적용하기 위한 수단을 제공한다. 앞으로는 역할 대 역할의 위임, 권한 대 역할 위임 등으로 연구를 확대하는 것이 필요하다.

참 고 문 헌

[1] Ravi Sandhu, Qamar Munawer, "How to do Discretionary Access Control Using Roles", Proceedings of 3rd ACM Workshop on Role-Based Access Control, Fairfax, Virginia, October 22-23, 1998.

[2] Longhua Zhang, Gail-Joon Ahn, Bei-Tseng Chu, "A Rule-Based Framework for Role-Based Delegation", Proceedings of 6th ACM Symposium an Access Control Models and Technologies, Chantilly, Virginia, May 3-4, 2001.

[3] Jonathan D. Moffett, Emil C. Lupu, "The Uses of Role Hierarchies in Access Control", Proceedings of 4th ACM Workshop on Role Based Access Control, George Mason University, Fairfax, VA. 1999.

[4] Moffett, J. D. "Control Principles and Role Hierarchies", Proceedings of 3rd ACM Workshop on Role Based Access Control, George Mason University, Fairfax, VA. 22-23 October 1998.

[5] Ravi Sandhu, Venkata Bhamidipati, "The ARBAC97 Model for Role-Based Administration of Roles : Preliminary Description and Outline", Proceedings of the second ACM workshop on Role-Based Access Control, pp. 41-50, Fairfax, Virginia, Nov. 6-7, 1997.

[6] Nataraj Nagaratnam, Doug Lea, "Secure Delegation for Distributed Object Environments", USENIX Conference on Object Oriented Technologies and Systems, April 1998.

[7] 심재훈, 박석, 역할기반 접근제어에 기초한 사용자 수준의 위임기법, 통신정보보호학회논문지 제10권 3호, pp. 49-62, 2000.9.

[8] J. Linn, M. Nystrom, "Attribute Certification: An Enabling Technology for Delegation and Role-Based Controls in Distributed Environments", ACM Workshop on RBAC, pp. 121-130, 1999.

[9] Ezedin barka, Ravi Sandhu, "Framework for Role-Based Delegation Model", Proceedings of 23rd National Information Systems Security Conference, pp. 101-114, Baltimore, Oct. 16-19, 2000.

[10] Ravi Sandhu, "Rational for the RBAC96

Family of Access Control Models", In Proceedings of 1st ACM Workshop on Role-Based Access Control, 1997.

[11] Ravi Sandhu, Venkata Bhamidipati, Qamar Munawer, "The ARBAC97 Model for Role-Based Administration of Roles", ACM Transactions on Information and System Security. Vol.2, No.1, pp. 105-135, 1999.

[12] Serge Abiteboul, Stephane Grumbach, "A Rule-Based Language with Functions and Sets", ACM Transactions on Database Systems, 16(1): 1-30(1991).

Rule-Based Framework for user level delegation model in Role Based Access Control

Chong-Hwa, Park*

Abstract

In current role-based systems, security officers handle assignments of users to roles. This may increase management efforts in a distributed environment because of the continuous involvement from security officers. The technology of role-based delegation provides a means for implementing RBAC in a distributed environment with empowerment of individual users. The basic idea behind a role-based delegation is that users themselves may delegate role authorities to other users to carry out some functions on behalf of the former. This paper presents a rule-based framework for user-level delegation model in which a user can delegate role authority by creating new delegation roles. Also, a rule-based language for specifying and enforcing the policies is introduced.

* Dept. of Software Semyung University