Chap 4.
Access Control

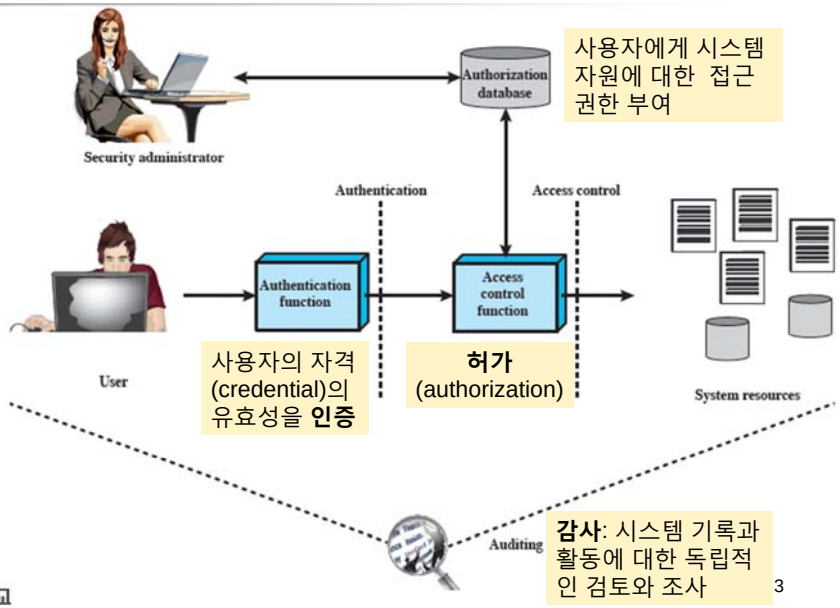
연세대학교

Access Control

- 접근 제어
  - 허가 받지 않은 방식으로 자원을 사용하는 것을 방지하는 것
  - 컴퓨터 보안에서 시스템 자원에 대한 접근 제어는 특히 중요함
- 접근 제어 정책
  - 어떤 상황에서
  - 누구(user)/무엇(process)에게
  - 어떤 유형의 접근이
    특정한 자원에 대해서 허용되는가

연세대학교

Access Control Principles



사용자에게 시스템 자원에 대한 접근 권한 부여

사용자의 자격 (credential)의 유효성을 **인증**

허가 (authorization)

감사: 시스템 기록과 활동에 대한 독립적인 검토와 조사

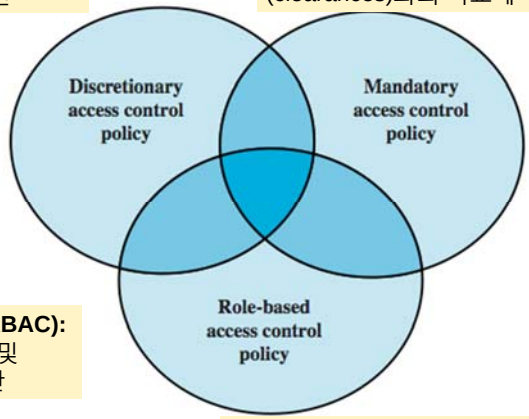
연세대학교

Access Control Policies

임의 접근 제어 정책 (DAC):
요청자의 신원과 사용자에 대한 접근 규칙에 기반

강제 접근 제어 정책 (MAC):
보안 label과 보안 허가증 (clearances)과의 비교에 기반

상호배타적이 아님



속성기반 접근 제어 (ABAC):
사용자의, 접근 자원, 및 현재 환경 조건에 기반

역할기반 접근 제어 정책 (RBAC):
사용자에게 할당된 역할과 역할에 허용된 접근 규칙에 기반

연세대학교

Access Control Basic Elements

subject
object를 접근할 수 있는 entity

- process 개념과 동일
- 일반적으로 그들이 처음 시행한 action에 대해서 책임을 짐
- 대개 세 클래스로 분류 : owner, group, world

object
접근이 제어되는 자원

- 정보를 포함하거나 받기 위해서 사용되는 개체 (레코드, 파일, 페이지, 프로그램, ..)
- 보호는 접근 제어가 작동하는 환경에 따라 다름

access right
subject가 object를 접근할 수 있는 방법

(예) read, write, execute, delete, create, search

Discretionary Access Control

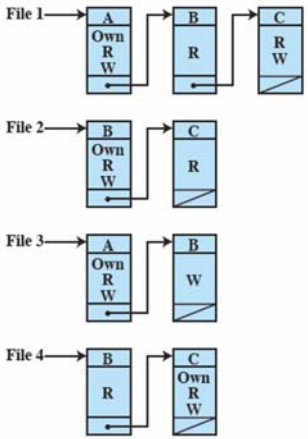
- DAC - 한 entity가 자신의 의지대로 다른 entity에게 어떤 자원을 접근할 수 있게 허용하는 접근권한을 승인 받을 수 있음.
- **접근 제어 행렬(Access Control Matrix)**
  - 한쪽 차원 – 자원 접근을 시도하는 식별된 subject로 구성
  - 다른 쪽 차원 – 접근되는 object들로 구성
  - 접근 행렬의 각 원소
    - 특정 object에 대한 특정 subject의 접근 권한을 나타냄

		OBJECTS			
		File 1	File 2	File 3	File 4
SUBJECTS	User A	Own Read Write		Own Read Write	
	User B	Read	Own Read Write	Write	Read
	User C	Read Write	Read		Own Read Write

(a) Access matrix

Access Control List

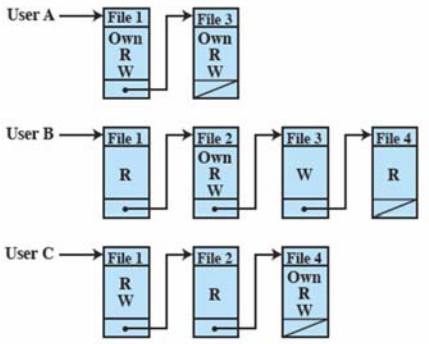
- 접근 제어 리스트(Access Control List: ACL)
  - 희소 접근 행렬에 대한 object 단위의 리스트 구현



(b) Access control lists for files of part (a)

Capability List

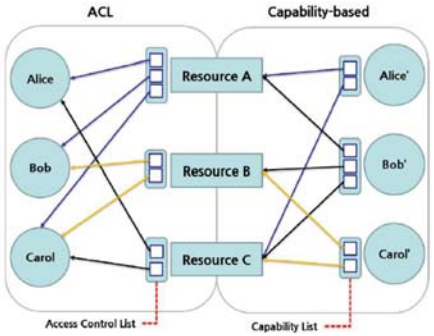
- 가용성(capability) 티켓 – 가용성 리스트(CL)
  - 접근 행렬에 대한 subject 단위의 리스트
  - 티켓이 사용자에게 분산됨 – 보안상 문제 발생 가능성
    - 티켓의 무결성이 보호되고 보장되어야 함



(c) Capability lists for files of part (a)

ACL-based and CL-based Access Control

- ACL 기반 접근 제어
  - 사용자가 자원에 대한 operation을 요청하는 경우
  - 권한 검증 : object가 갖고 있는 ACL을 통해
- CL 기반 접근 제어
  - 사용자가 갖고 있는 CL을 서비스 제공자에게 제시
  - 권한 검증 : 제시한 capability를 확인



Authorization Table

- 권한 테이블
  - 한 행에 하나의 object에 대한 한 subject의 한 가지 접근권한을 포함
  - Subject 기준 정렬
    → capability list
  - Object 기준 정렬
    → ACL
  - **관계형 데이터베이스**는 이러한 형태의 권한 테이블을 쉽게 구현 가능

Subject	Access Mode	Object
A	Own	File 1
A	Read	File 1
A	Write	File 1
A	Own	File 3
A	Read	File 3
A	Write	File 3
B	Read	File 1
B	Own	File 2
B	Read	File 2
B	Write	File 2
B	Write	File 3
B	Read	File 4
C	Read	File 1
C	Write	File 1
C	Read	File 2
C	Own	File 4
C	Read	File 4
C	Write	File 4

Access Control Model

- DAC를 위한 일반적 모델 – object 범위를 확장
  - 이 모델에서 subject 집합, object 집합, object에 대한 subject의 접근을 통제하는 규칙 집합을 가정
  - 시스템 보호 상태의 표현
    - 특정 시점에서 각각의 object에 대한 subject의 접근 권한을 명시하는 정보의 집합으로 표현
  - 보호 상태를 나타내기 위해 ACL에서 object 분야를 확장함
    → 확장 접근제어 행렬
    - Process : delete, stop, wakeup 권한
    - Device : read, write, control(예: seek), block/unblock
    - Memory Location: 특정 위치에 대한 read/write 접근 제어
    - Subject : 다른 subject에 대한 object 접근 권한을 승인, 삭제

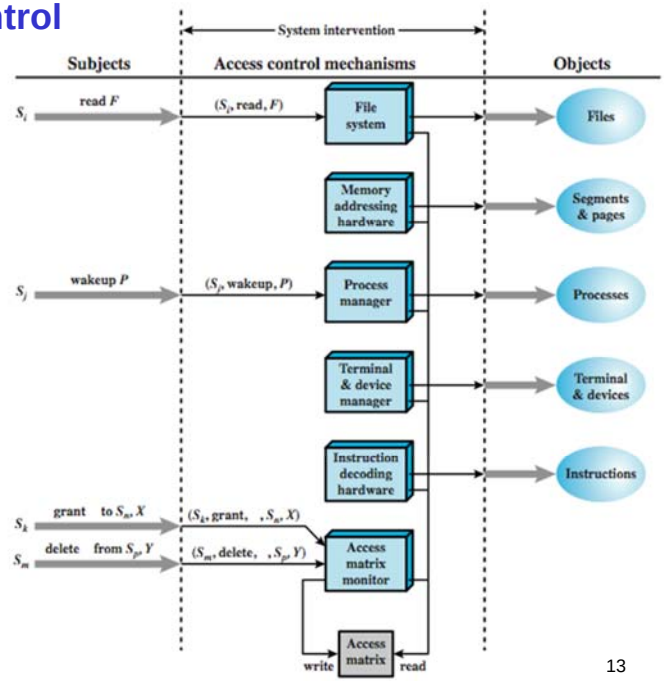
Extended Access Control Matrix

- DAC를 위한 일반적 모델 – object 범위를 확장
  - Process : delete, stop, wakeup 권한
  - Device : read, write, control(예: seek), block/unblock
  - Memory Location: 특정 위치에 대한 read/write 접근 제어
  - Subject : 다른 subject에게 object 접근 권한을 양도, 승인, 삭제

		OBJECTS								
		subjects			files		processes		disk drives	
		S_1	S_2	S_3	F_1	F_2	P_1	P_2	D_1	D_2
SUBJECTS	S_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
	S_2		control		write *	execute			owner	seek *
	S_3			control		write	stop			

* - copy flag set

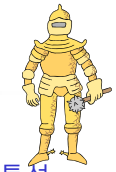
Access Control Function



Access Control System Commands

Rule	Command (by S_0)	Authorization	Operation
R1	**transfer** $\begin{Bmatrix}\alpha^* \\ \alpha\end{Bmatrix}$ **to** S, X	'α^*' in $A[S_0, X]$	store $\begin{Bmatrix}\alpha^* \\ \alpha\end{Bmatrix}$ in $A[S, X]$
R2	**grant** $\begin{Bmatrix}\alpha^* \\ \alpha\end{Bmatrix}$ **to** S, X	'owner' in $A[S_0, X]$	store $\begin{Bmatrix}\alpha^* \\ \alpha\end{Bmatrix}$ in $A[S, X]$
R3	**delete** α **from** S, X	'control' in $A[S_0, S]$ or 'owner' in $A[S_0, X]$	delete α from $A[S, X]$
R4	$w \leftarrow$ **read** S, X	'control' in $A[S_0, S]$ or 'owner' in $A[S_0, X]$	copy $A[S, X]$ into w
R5	**create object** X	None	add column for X to A; store 'owner' in $A[S_0, X]$
R6	**destroy object** X	'owner' in $A[S_0, X]$	delete column for X from A
R7	**create subject** S	none	add row for S to A; execute **create object** S; store 'control' in $A[S, S]$
R8	**destroy subject** S	'owner' in $A[S_0, S]$	delete row for S from A; execute **destroy object** S

Protection Domains



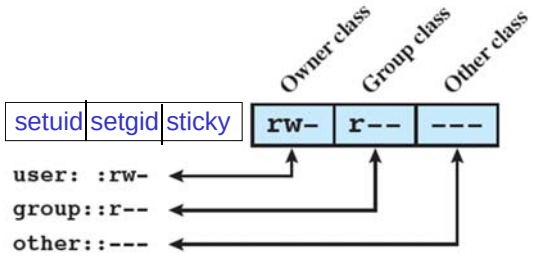
- 해당 object에 대한 접근 권한과 함께 존재하는 object 집합
- 보호 도메인을 capability와 연관시켜 사용할 때에 더 많은 융통성
  - 사용자는 접근권한의 부분 집합으로 프로세스를 생성할 수 있음
  - 새로운 보호 도메인이 정의됨
- 프로세스와 보호 도메인의 관계는 정적 또는 동적일 수 있음
  - 프로시저에 따라서 다른 접근 권한 요구 가능
- 보호 도메인의 예 – user mode 와 kernel mode
  - user mode : 특정 영역의 메모리 보호, 특권 명령어 실행 불가
  - kernel mode : 메모리 보호 영역 접근 및 특권 명령어 실행 가능

UNIX File Access Control

- UNIX file은 inode (index node)를 사용하여 관리됨
  - inode는 특정 파일에 필요한 핵심정보를 갖는 제어 구조
  - 여러 개의 파일 이름이 단일 inode와 연관될 수 있음
  - active inode는 정확히 한 개의 파일과 연관됨
  - 파일속성, 허가 및 제어 정보는 inode에 저장됨
  - 디스크에 파일시스템에 있는 모든 파일의 inode를 포함한 inode table이 존재
  - 파일이 open될 때에, 파일의 inode를 주메모리의 inode table에 저장함
- 디렉토리는 계층적 트리 구조를 가짐
  - 디렉토리는 파일과 다른 디렉토리들을 포함할 수 있음
  - 파일 이름과 연관된 inode에 대한 포인터를 포함

UNIX File Access Control

- 유일한 사용자 ID (식별자)
- 특정 그룹에 속함
- owner ID, group ID 및 protection bit는 inode의 일부
- 12개의 protection bit
  - 9 bit는 owner, group, other의 read/write/execute 권한을 지정
  - 3 bit는 파일이나 디렉터리의 특수 동작을 정의



(a) Traditional UNIX approach (minimal access control list)

Traditional UNIX File Access Control

- "set user ID"(SetUID)
  - 파일을 실행할 때에 일시적으로 그 파일을 생성한 사용자의 권한을 사용자에게 부여
- "set group ID"(SetGID)
  - 파일을 실행할 때에 일시적으로 그 파일을 생성한 그룹의 권한을 사용자에게 부여
- sticky 비트
  - 파일에 설정할 때에 – 파일이 실행된 후 파일 내용을 메모리에 유지해야 함
  - 디렉터리에 적용할 때 – 디렉터리에 있는 파일의 소유자 만이 파일 이름변경, 이동, 삭제 가능
- superuser
  - 일반적인 접근 제어에서 제외됨
  - 시스템 전체에 대한 접근 가능

Access Control Lists (ACLs) in UNIX

- modern UNIX systems은 ACL을 지원
  - FreeBSD, OpenBSD, Linux, Solaris
- FreeBSD
  - setfacl 명령어를 사용하여 UNIX 사용자 ID와 그룹 목록을 지정
  - 임의의 개수의 user와 group이 파일과 연관될 수 있음
  - 보호 비트: 읽기, 쓰기, 실행
  - 파일에 ACL이 있을 필요는 없음.
  - 파일에 **확장된 ACL**이 있는지 여부를 나타내는 추가적인 보호 비트가 포함됨

Access Control Lists (ACLs) in UNIX

- 프로세스가 파일 시스템 object에 대한 접근 요청하면 2 단계 수행
- 1단계 : 요청하는 프로세스에 가장 일치하는 ACL 항목 선택
  - ACL 항목은 소유자, 지명된 사용자, 그룹, others 등 단일 항목만 접근을 결정
- 2단계 : 일치하는 항목이 있는지 확인
  - 선택된 항목이 충분한 승인 정보를 포함하고 있는지 검사
  - 프로세스는 둘 이상의 그룹에 속할 수 있음
  - 프로세스는 한 개 이상의 그룹 멤버일 수 있기 때문에 한 개 이상의 그룹 항목이 일치될 수 있음

Mandatory Access Control (MAC)

Top Secret

Secret

Confidential

Unclassified

Labeling Mechanism이 사용됨

다단계 보안 강화를 통하여 불법적인 정보유출 방지

Military Security

subject와 object를 보안 수준에서 엄격하게 분류 필요

단점: 너무 엄격, 융통성이 없음

'secret' 사용자는 'top secret' 파일을 읽지 못함

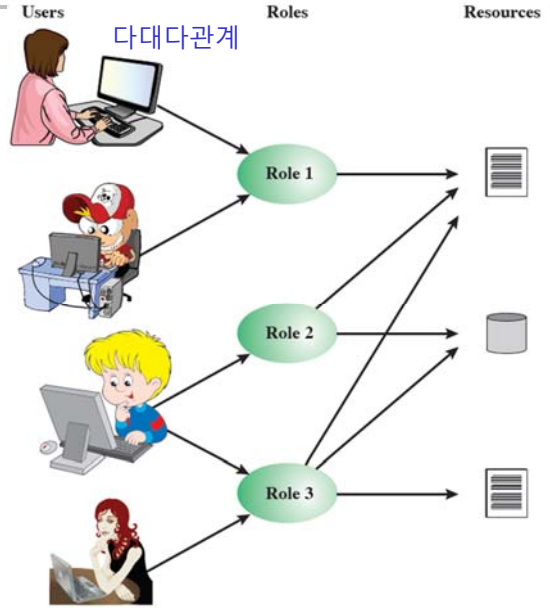
Compartments and Sensitivity Levels

- 정보 접근은 알 필요 정도에 의해서 제한됨
- 구획(Compartment):
  - 분류된 각 정보는 구획이라고 하는 하나 이상의 프로젝트와 연관될 수 있음.

Role-Based Access Control (RBAC)

- 전통적인 DAC 시스템
  - 개별 사용자 및 그룹의 접근 권한을 정의하는 RBAC모델
  - 어떤 기관 내의 작업 기능(job function)으로 **역할**을 정의
- RBAC 시스템
  - 각각의 사용자 대신에 **역할(role)**에 접근 권한을 할당
  - 사용자는 정적/ 동적으로 각자 책임에 따라 다른 역할에 할당됨
  - 상업적으로 널리 쓰이고 연구가 활발히 진행 중

Users, Roles, Resources



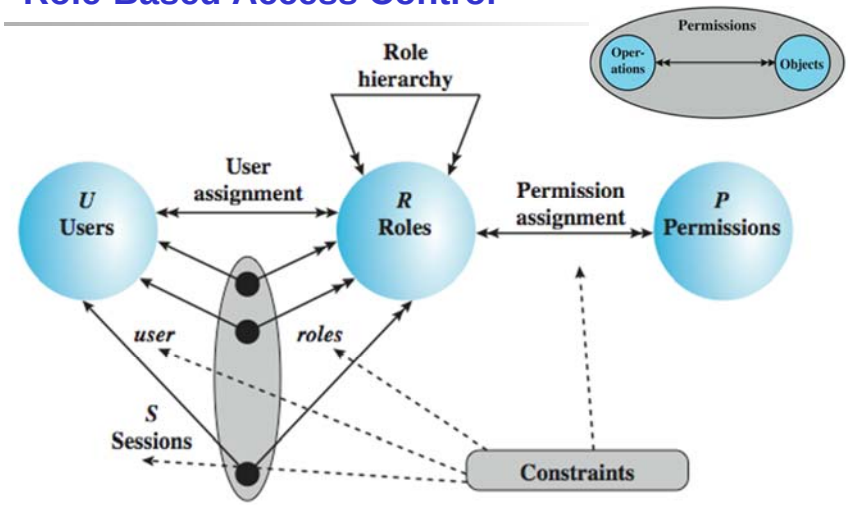
Access Control Matrix of RBAC

	R_1	R_2	...	R_n
U_1	✕			
U_2	✕			
U_3		✕		✕
U_4				✕
U_5				✕
U_6				✕
...				
U_m	✕			

OBJECTS

ROLES	R_1	R_2	R_n	F_1	F_1	P_1	P_2	D_1	D_2
R_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
R_2		control		write *	execute			owner	seek *
...									
R_n			control		write	stop			

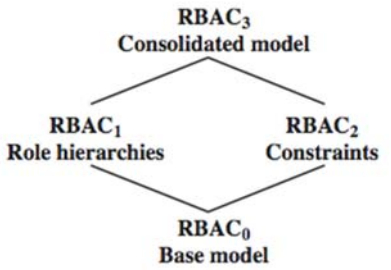
Role-Based Access Control



- 그림 설명
  - 실선 – 관계 또는 매핑을 의미
    - 하나의 화살촉 : 1개
    - 두 개의 화살촉 : 다수
  - 세션 – 사용자와 사용자에게 할당되어 있는 1개 이상의 역할 사이의 일시적인 "1대 다" 관계
  - 사용자 – 특정 작업에 필요한 역할에 대해서만 세션을 만듦
- 접근제어에 대한 융통성(flexibility)과 세분성(granularity) 제공
  - 사용자-역할과 역할-승인(Permission)과의 다-대-다 관계 (전통적인 DAC 구조에서 찾아 볼 수 없음)
  - DAC 구조에서는 사용자가 자원에 대해서 필요 이상의 접근 권한을 승인 받을 수 있는 위험이 존재

RBAC Models

- RBAC의 다양한 측면을 분류하기 위해 RBAC을 기능별로 추상 모델 집합을 정의
- RBAC 참조 모델



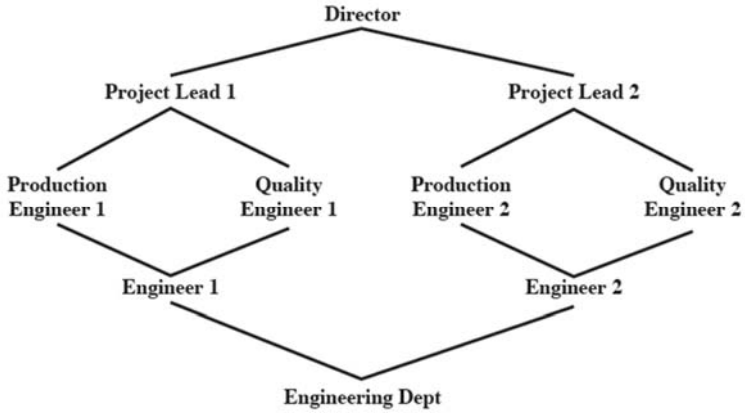
Models	Hierarchies	Constraints
$RBAC_0$	No	No
$RBAC_1$	Yes	No
$RBAC_2$	No	Yes
$RBAC_3$	Yes	Yes

RBAC model

- **RBAC0** – 기본 모델
  - RBAC 시스템의 최소한의 기능만 포함
- **RBAC1** – 역할 계층 구조
  - RBAC0의 기능과 하나의 역할이 다른 역할에 승인 정보를 상속할 수 있는 역할 계층구조 추가
- **RBAC2** – 제약
  - RBAC0의 기능 + RBAC 시스템의 구성 요소를 수정할 수 있는 방법에 대한 제약 조건 추가
- **RBAC3** – 통합 모델
  - RBAC0, RBAC1, RBAC2 의 기능을 모두 포함

Example of Role Hierarchy (RBAC1)

- 보통보다 큰 책임을 가진 직무는 자원에 접근할 수 있는 더 큰 권한을 가짐



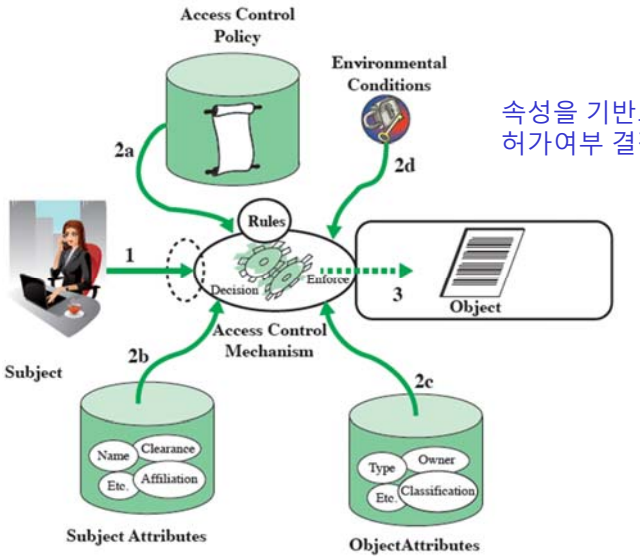
Constraints – RBAC2

- RBAC을 조직 내의 관리 및 보안 정책의 세부 사항에 적응하기 위한 수단을 제공
  - 제약: 역할과의 관계 또는 역할과 관련된 상태의 관계로 정의
- 제약 조건
  - 상호 배타적 역할 – 역할 집합에 있는 하나의 역할에만 할당
  - cardinality – 역할에 대한 최대수
  - 필요조건 – 최소한의 권한 개념의 구현을 구조화하는 데 사용

Attribute Based Access Control (ABAC)

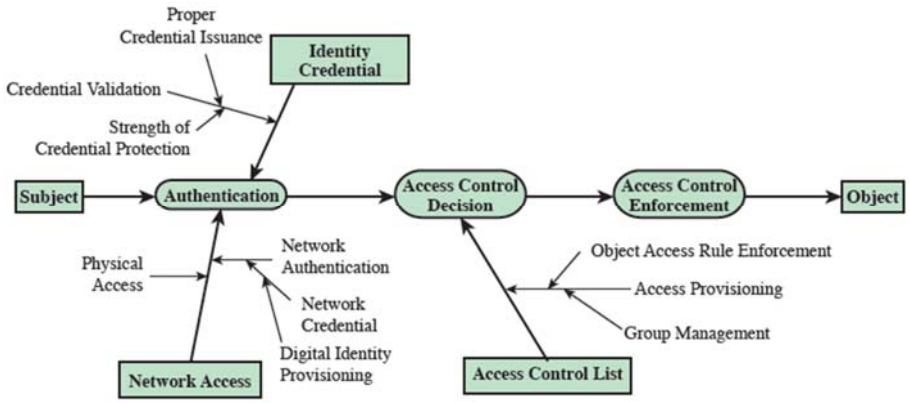
- ABAC 모델 – 허가(authorization)를 자원과 subject의 성질(속성)에 대한 조건을 표현하여 정의함
- 속성(attribute)
  - subject, object, 환경 조건, 권한에 의해 미리 정의되고 할당된 요구되는 동작의 특정 측면을 정의
- ABAC 모델의 세 가지 유형의 속성
  - **subject 속성** : 정보가 object 사이를 이동하게 하거나 시스템 상태를 변경하도록 하는 능동적 entity (사용자, 프로세스의 속성)
  - **object 속성** : 정보를 포함하거나 받는 수동적 정보시스템 관련 entity (MS워드 문서의 속성 등)
  - **environment 속성** : 운영, 기술, 상황 또는 정보접근이 발행하는 환경 (현재 날짜, 시간, 네트워크보안레벨 등)

ABAC Scenario

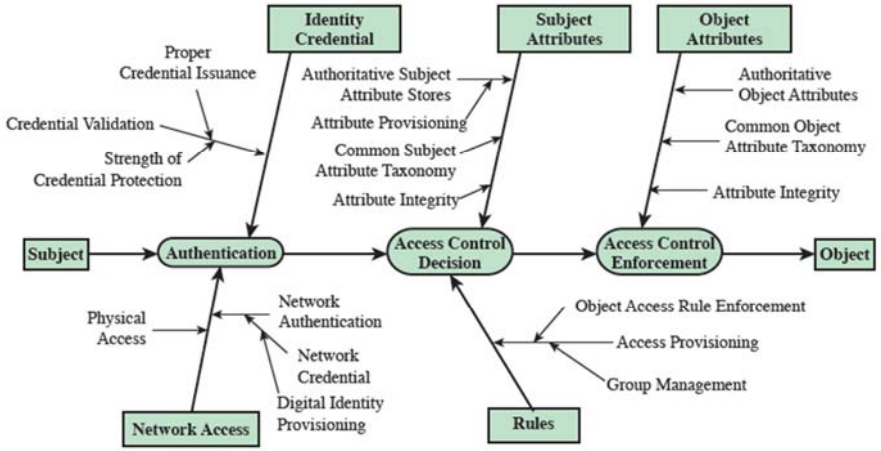


속성을 기반으로
허가여부 결정

ACL Trust Chain (NIST)



ABAC Trust Chain (NIST)



신뢰의 root가 ACL에 비해서 ABAC가 많다.
object의 소유자가 제어하지 못하는 많은 소스에서 유래함