유저 권한관리 모델 유형

용어 정의

a. 접근 제어 모델에서 인가를 받으려는 사용자를주체(Subject) 라고 합니다.
- 주체는 특정 사용자 개인이 될 수도 있고, 특정 사용자 그룹이 될 수도 있으며, 또는 전체 사용자를 가리킬 수도 있습니다.

b. 객체(Object)는 사용자가 접근하고자 하는 서비스나 자원, 정보와 같은 접근 대상을 말합니다.
- 네이버 카페의 게시판이 객체의 예입니다.
- 권한(Permission)은 객체에 허용된 주체가 수행할 수 있는 행위의 목록을 말합니다. 보통권한에는 읽기, 쓰기, 생성, 삭제, 실행, 검색 등과 같은 권한이 존재할 수 있습니다.

c. 결국 접근 제어(Access Control)라는 것은객체(Object)에 접근하고자 하는주체(Subject)가 해당 객체(Object)에 접근할 수 있는 권한(Permission)을 가지고 있는지 판단하는 것입니다.

1. 모델 목록

1.1 강제적 접근 제어 모델 (Mandatory Access Control, MAC)

1.2 임의적 접근 제어 모델 (Discretionary Access Control, DAC)

1.3 역할 기반 접근 제어 모델 (Role Based Access Control, RBAC)

1.4 속성 기반 접근 제어 모델 (Attribute Based Access Control, ABAC)

1.1 강제적 접근 제어 모델 (Mandatory Access Control, MAC)

- 미리 정해진 정책과 보안 등급에 의거하여 주체(Subject)에게 허용된 접근 권한과 객체(Object)에 부여된 허용 등급을 비교하여 접근을 제어하는 방법
- 강제적 접근 제어 모델에서는 높은 보안 수준을 요구하는 정보는 낮은 보안 수준을 가진 주체가 접근할 수 없습니다.
    - 시스템 관리자가 이러한 권한 수준을 제어하며, 사용자들은 어떠한 접근 권한 수준도 설정할 수 없게 됩니다.
    - 객체의 소유자라고 할지라도 그 객체에 접근할 수 있는 보안 등급을 부여받지 못하면 그 객체에 접근할 수 없습니다.

1.1 강제적 접근 제어 모델 (Mandatory Access Control, MAC)

예시 : 네이버나 다음 카페에서 회원 등급에 따라 접근할 수 있는 게시판과 없는 게시판을 구분하는것 입니다.

각 게시판(객체)마다 접근할 수 있는 허용 등급이 정해져 있기 때문에 객체와 주체 사이의 권한 등급을 직접 비교하여 접근 허용 여부를 결정하는 것 입니다..

등업 시스템을 생각하면 좋습니다.

1.1 강제적 접근 제어 모델 (Mandatory Access Control, MAC)

- **장점** : 강제적 접근 제어 모델은 중앙 집권적인 관리자에 의해 제어되기 때문에 보안성이 높은 편이어서 주로 군(military)이나 방화벽처럼 강력한 보안이 필요한 곳에서 주로 사용합니다.
- **단점** : 권한 관리 기능이 단순하고 제한적이어서 주체별로 접근 제어를 다르게 적용할 수 없으며, 모든 주체와 객체에 일일이 허용 등급 설정을 해주어야 하므로 설정이 복잡합니다.

1.2 임의적 접근 제어 모델 (Discretionary Access Control, DAC)

- 객체에 대한 접근을 사용자나 그룹의 신분을 기준으로 제한하는 방법
  - 객체의 소유자는 다른 주체에 대해 객체에 대한 접근 권한을 설정하는 방식입니다.
- 여기서 임의적(Discretionary)이라는 말은 객체 소유자의 판단에 따라 권한을 줄 수 있다는 뜻입니다.
- DAC 을 구현하는 방법은 대표적으로 다음 4가지의 구현 방법이 있습니다. (이것에 대한 구현방법은 내용이 너무 길어지므로, 종류만 소개할께요)
  - 접근 제어 행렬 (Access Control Matrix)
  - 접근 제어 목록 (Access Control List: ACL)
  - 가용성 티켓 (Capability Tickets)
  - 권한 테이블 (Permission Table)

1.2 임의적 접근 제어 모델 (Discretionary Access Control, DAC)

예시 : 리눅스 파일이나, 디렉토리에 대해 접근제어 방식을 말합니다.

이 유저의 권한은 RWX, 이 유저의 권한은 R.. 등.

파일은 파일의 소유자가 그 파일에 대한 접근 제어를 설정할 수 있습니다.

1.2 임의적 접근 제어 모델 (Discretionary Access Control, DAC)

- **장점** : 주체와 객체를 알고 있으면 어떤 권한을 가지고 있는지 바로 알아낼 수 있습니다.
- **단점** : 주체의 수나 객체의 수가 많아지면 쓸데없이 사용되는 메모리 공간이 많아지는 단점을 가지고 있습니다.
  어느 한 객체에 대해 많은 사용자가 권한을 부여받았을 경우에는 리스트가 길어져 탐색이 오래 걸리게 됩니다.

1.3 역할 기반 접근 제어 모델 (Role Based Access Control, RBAC)

- 권한을 사용자 개인이 아닌 역할 그룹에 부여하고, 사용자에게 역할을 할당하여 접근 제어를 하는 방식
- 이름 그대로 역할에 따라 자원에 대한 접근을 제어하는 방식입니다.
    - 대부분의 상용 플랫폼이 지원하는 권한 관리 체계로 권한 영역에 역할이라는 개념을 도입해 표현력을 확장한 방식입니다.
    - 임의적 접근 제어(DAC)과 강제적 접근 제어(MAC)의 단점을 보완한 방식으로써 사용자에게 정적 혹은 동적으로 역할 그룹을 할당할 수 있습니다.
        - 이는 사용자 정보는 자주 바뀌어도 역할 정보는 자주 바뀌지 않는다는 것에 착안한 모델이죠.
- 일반적으로 직무를 기반으로 하는 서비스들에서 많이 사용되는 접근 제어 방법으로 강제적 접근 제어(MAC)과 비슷하게 각 주체에 허용된 접근 수준(Clearance)과 객체에 부여된 허용등급(Classification)에 근거하여 접근 제어가 이루어집니다.
- 사용자와 역할 사이의 관계는 보통 다대다(N:M) 관계이기 때문에 한 사용자가 여러 개의 역할을 부여받을 수 있습니다.
    - 역할과 객체 사이의 관계도 다대다(N:M) 관계이기 때문에 한 역할이 여러 객체에 대한 접근 권한을 부여받을 수 있습니다.

1.3 역할 기반 접근 제어 모델 (Role Based Access Control, RBAC)

실제로 많은 서비스들이 회원 가입과 탈퇴는 빈번하게 일어나지만, 그 회원들이 부여받을 수 있는 역할의 종류와 역할과 객체 사이의 권한 관계는 잘 변하지 않습니다.

예시 : 이 유저는 이런이런 페이지를 볼수있는 기능을 가진 뷰어 권한을 가지고 있게 합니다.

그럼 유저의 탈퇴와는 상관없이 역할기반 권한이 존재하는거고, 사용자는 그걸 받아갑니다.

1.3 역할 기반 접근 제어 모델 (Role Based Access Control, RBAC)

- 역할 정의

```
{
  "name": "roles/testRole",
  "title": "billing role",
    "includedPermissions": [
    "paymentStatement.list",
    "usage.list"
    ]
  }
```

- 역할 부여

```
{
  "bindings": [{
  "role": "roles/testRole",
    "members": [
    "user:viewrain@hohoho.com"
    ]
  }]
  }
```

1.3 역할 기반 접근 제어 모델 (Role Based Access Control, RBAC)

- **장점** : 권한 관리자는 다수의 사용자에 대해 일일이 접근 권한을 관리하지 않아도 되고 적절한 역할만 부여해주면 되기 때문에 권한 관리 부담이 줄어듭니다.
- **단점** : 다양한 권한 요소를 고려하다 보면 권한 조건이 늘어날 때마다 역할의 개수가 과도하게 늘어날 수 있는 단점이 있으므로 역할을 개수를 적절히 유지하는 것이 중요합니다.

1.4 속성 기반 접근 제어 모델 (Attribute Based Access Control, ABAC)

- 객체와 주체의 속성에 대한 조건을 기술하여 접근 제어를 하는 방식
    - 어떤 객체에 접근하기 위해 만족시켜야 하는 속성에 대해 정의하고, 그 객체에 접근하려는 주체가 그 속성을 가지고 있는지를 검사하여 접근 제어를 수행합니다.
    - 속성은 주체 이름, 자원 유형, 현재 시간 등 다양하게 존재하며 지원하는 속성의 종류는 각 플랫폼마다 상이하다고 합니다.

예시 : 예를 들어 파일 1에 접근하기 위해서는 사용자의 type 속성에 admin이라는 태그가 달려 있어야 한다고 정의할 수 있습니다.

1.4 속성 기반 접근 제어 모델 (Attribute Based Access Control, ABAC)

- 속성 기반 접근 제어 적용

```
{
  "bindings": [{
      "role": "roles/testRole",
      "members": [
          "user:viewrain@hohoho.com"
          ]}
          ],
  "condition": {
  "title": "DateTime Expires",
  "description": "Expires at noon on 2023-12-31 UTC",
  "expression": "request.time < timestamp('2022-06-01T12:00:00Z')"
  }
```

1.4 속성 기반 접근 제어 모델 (Attribute Based Access Control, ABAC)

- **장점** : 시스템의 다양한 요소를 반영할 수 있기 때문에 표현력과 유연성이 좋습니다.
- **단점** : 큰 규모의 시스템에서는 일일이 속성을 적용하기 어렵고 각 객체 접근마다 복잡한 속성 조건을 계산해야 하기 때문에 성능이 다소 느립니다. (혹은 느려질 수 있습니다.)

결론

- 결국 완벽한 접근 제어(사용하기도 쉽고 성능도 나쁘지 않은)를 하기 위해서는 그 시스템을 사용하는 사용자가 누구인지, 그리고 보호하려는 객체는 어떤 것이 있으며, 각 객체 사이의 관계와 특징은 무엇인지, 어느 정도 복잡한 접근 제어까지 지원해야 하는지 등의 요구 사항을 명확히 파악하는 것이 무엇보다도 중요하다고 할수 있습니다.

- 인터넷에 권한관리에 대한 내용을 검색해보면, 큰 틀이지만 공통된 방안을 제시하고는 있습니다.
  1. 미리 제공되는 역할을 사용할 것
  2. 서비스별 계정을 분리할 것
  3. 최소 권한만 부여할 것