**Chapter 1: Computer and Network Security Concepts**

**NIST가 정의한 Computer Security:**

Integrity, availability, confidentiality를 보존하기 위해 자동화된 information system을 보호하는 것

**Computer security 의 목적:**

* Confidentiality (기밀성):
  + Data confidentiality: 허가 받은 사람만 중요한 정보를 볼 수 있어야 한다.
  + Privacy: 개인들이 자신의 정보를 control 할 수 있음. (ex. 회원가입 할 때 동의 절차) 아무나 임의의 방식으로 개인의 정보를 배포할 수 없음.
* Integrity (무결성):
  + Data integrity: 정보, 데이터, 프로그램이 미리 허가된 방법으로만 변경될 수 있음.

변경되었을 때 변경사실을 확인 가능 해야함.

* + System integrity: 어떤 시스템이 허가 받지 않은 방식으로 바뀌었을 때, 막거나 확인할 수 있어야함. (허가 받은 사람은 허가 받은 방식으로 변경 가능)
* Availability (가용성):

허가 받은 유저에게는 정상적으로 시스템이 작동 해야함.

**<보안의 요구 조건>**

5가지 용어 기억하기: CIA(Confidentiality,Integrity, Availability) +AA(Accountability, Authenticity)

Accountability: 책임 소재를 가질 수 있는 성질

Authenticity: 인증 (누가 진짜인지 인증)

**CIA의 대표적인 적용 사례**

* Confidentiality : 학교 전산 시스템의 학생 성적 정보 ( 아무한테나 공개되면 안됨)
* Integrity: 병원 시스템의 환자마다 투약하면 안되는 약 정보 (알러지).
* Availability: 어떤 시스템이 다른 시스템의 연결경로로써 쓰이는 경우.

Ex) 본인 인증 시스템 -> 작동하지 않는다면, 이후의 결제와 같은 것이 진행되지 않음 (이러한 시스템은 항상 가용한 상태를 유지해야함)

**컴퓨터 보안의 장애물 (**공격자 vs 방어자의 싸움, 공격자가 항상 유리**)**

* 간단하지 않음

방어자는 potential한 공격을 모두 고려하여 방어해야 하지만, 공격자는 한가지 방법만 선택 가능

* Constant monitoring: 항상 지켜봐야함. (빈 틈이 없어야함)
* 여러가지 알고리즘을 조합해야함.
* 보안에 대한 투자가 little benefit을 가져올 수 있다. (투자 대비 효용성이 낮을 수 있다.)
* Strong security는 효율성을 낮추는 장애물이 될 수도 있다. (안정성 vs 편의성)

**OSI Security Architecture**

OSI에서 제안한 security architecture: (간단히 설명)

* Security attack
* Security mechanism
* Security service

**Security attack**

* Threat(위협) vs attack(공격) : potential vs 실제로 실현된 것
* **Passive attack**: 시스템의 상태가 변하지 않는 것
  + The release of message contents

Ex) Bob --------(Darth가 중간에서 도청)---------🡪 Alice

(Alice는 자신이 피해 당했다는 사실을 모를 수 있음)

해결 방법: 암호화로 해결

* + Traffic analysis

암호화된 내용이라도, 정보의 주고받음 (traffic)을 분석하여 정보를 얻어낼 수 있음

해결 방법: 트래픽 자체를 숨길 수 있음 ex) Tor(익명 망)

* **Active attack**: 시스템/자원을 변형시키는 것
  + Masquerade

Ex) (Bob은 아무 것도 안 보냈는데, Darth가 보냄)------🡪 Alice

(Bob인 척 하는 것)

* + Replay

한번 보낸 것을 복사해놨다가 계속 보내는 것

Ex) (bob->elice 만원 송금) X100

* + Modification of messages

Ex) Bob ------(Darth가 중간에서 변조해서 보냄)------🡪 Alice

* + Denial of service (서비스 거부 공격)

Ex) Bob ------(Darth가 중간에서 가로채서 막음)------🡪 Alice

Client가 server에 접속할 때, client가 서버로부터 정상적인 서비스를 받지 못하도록 막는 것.

**Security Services**

* Data confidentiality:
  + passive attack에 대한 방어
  + 내용 뿐 아니라, traffic(source destination, frequency, length,,,)도 보호해야함.
* Data Integrity:
  + Stream of message 일 경우 (Connection-oriented integrity service)

순서 중요 O -> 메시지 변조 + reordering , replays 되었는가.

* + Single message일 경우 (Connectionless integrity service)

순서 중요 X -> 메시지가 변조(modification) 되었는가. (duplication, insertion, modification, 삭제)

* Availability:

Denial-of-service attack (DOS 공격)에 대응

* Authentication:
  + Peer entity authentication : 사용자 인증 (주체확인)
  + Data origin authentication: 신원을 가장하여 공격하는 것을 막는 것
* Accountability: 문제가 생겼을 때 책임 소재를 밝힐 수 있다. (넓은 개념임)
  + Nonrepudiation: (repudiation:부인) -> 부인하지 못하도록 하는 것

B -----(전송)------🡪 A

A는 받은 사실을 부인할 수 없다. B는 전송사실을 부인할 수 없다.

* Access Control (접근 제어): 위의 5가지 속성을 총 망라하는 개념

어떤 시스템, 자원, 애플리케이션에 대해서 접근을 한정시키고 제어할 수 있는 능력

달성하기 위해서는 각 entity들의 신원 인증 절차 먼저 필요.

<정리>

|  |  |  |  |
| --- | --- | --- | --- |
| service | attack |  | Security mechanism |
| Confidentiality  (기밀성) | Release | Passive attack | Encryption(암호화)/traffic 보호하는 별도의 방법 필요 |
| Integrity  (무결성) | Modification (변조) | Active attack | 무결성과 인증 제공하는 방법 : message authentication code(MAC), digital signature(전자서명)  digital signature(전자서명): 무결성, 인증, nonrepudiation(부인방지)  MAC: 무결성, 인증 |
| availability | Dos 공격 |  |
| Authentication | Masquerade (가장) |  |

**Fundamental Security Design principles ( 2가지 설명 하시오)**

* Economy of mechanism
  + 간단하고 최소한 작게 만들어야 함 (비용적인 측면, 관리적인 측면)

가능하면 요구조건을 만족하는 범위내에서 간단하게.

* + 복잡하게 디자인하면 공격자가 취약점을 발견할 수 있는 가능성이 많아짐.
* Fail-safe defaults (중요)
  + Decisions에 엑세스하는 방법은 Exclusion이 아닌, permission에 기반해야함.
  + Default 상황: 엑세스가 별로 없는데, 어떤 상황에서 엑세스가 허용될 것인가 명시

(대부분을 허용하고 안되는 경우를 명시 X)

* + 대부분 파일 엑세스 시스템과 많은 애플리케이션은 이러한 fail-safe defaults을 사용함.
* Complete mediation
  + 모든 엑세스가 access control mechanism에 의해서 확인되어야함.
* Open design
  + 보안 시스템을 다 열어놓고 누구나 볼 수 있게 설계

닫아 놓고 일부의 사람들만 설계를 참여한다면, 설계에 참여한 일부의 사람들의 지식만을 활용을 하게됨 -> 미처 생각하지 못한 취약점이 생길 수 있음.

* + - * 아얘 열어놓고 다른 사람들에게 취약점을 확인해 줄 것을 요청

여러 사람의 검증을 통할 수 있음

Ex) AES (미국 표준 암호): 개방설계의 사례

* Separation of privilege
  + 권한을 세분화 시킨다. (한 가지 권한으로 한 가지 일만 할 수 있게 or 한 가지 일을 하기 위해 여러 권한이 필요하도록)

이유: 하나가 뚫렸을 때 전체가 망가지지 않도록

Ex) Multifactor user authentication : password 와 smartcard 두 개다 통과해야함

(2가지 privilege 만족해야함)

* Least privilege
  + 권한을 부여할 때, 그 작업에 필요한 최소한의 권한만 부여하라

Ex) 사용자가 파일을 읽으려고 할 때, 구지 write 권한까지 주지 마라

* Least common mechanism
  + 서로 다른 사용자들에 의해서 공유되는 함수를 최소화하여 디자인한다.

이것 저것 공유하다보면 의도치 않게 다른 사용자에게 유출이 될 수 있음.

* Psychological acceptability
  + 사용자들을 필요 이상으로 불편하게 하지 마라.
  + 사용자 입장에서는 security mechanism이 있는지 없는지 모르게 하는게 가장 좋음. (편리성)
* Isolation
  + Public access system과 critical resource를 다루는 system은 서로 분리되어야 된다.

Ex) 발전기들을 제어하는 제어망 (critical resource) , 일반적인 용도로 쓰는 네트워크(public system), 홍보용도로 쓰는 공공이 접근할 수 있는 시스템(public system) -> 분리해야한다.

* + Process 와 file간에도 엑세스가 분리되어야함.
* Encapsulation
  + Object-oriented functionality에 기반한 특정한 형태의 isolation으로 볼 수 있음.

밖에서는 필요한 기능에 대해서만 접근하고, 그 안에 내부적인 구성은 외부에서 볼 수 없다.

* + 데이터 보호할 수 있음
* Modularity
  + 모듈 별로 설계해서 한 쪽 부분의 오류가 다른 쪽으로 전파되지 않도록 분할
* Layering
  + 여러 개 (혹은 겹치는) protection 방법들을 도입하자. (겹치는 것도 가능)

(separation of privilege와 비슷)

* Least astonishment
  + 사용자를 놀라게 하면 안됨
  + 사용자가 상식적으로 기대할 수 있는 방식으로 동작해야함

**Attack Surfaces : 공격자가 뚫고 들어갈 수 있는 표면들**

시스템에서 활용가능한 취약점들

Ex)

어떤 웹이나 다른 서버 상에서 돌고있는 프로그램들이 연 열린 포트들이 있고, 밖에서부터 그 포트를 통해 무언가가 들어오기를 기다리고 있을 때,

필요 없는 포트를 잘못해서 열어두게 되면, 원하지 않는 외부의 접근이 그 포트를 통해 들어올 수 있음. (가능한 취약점)

* Network attack surface

네트워크를 통해서 공격이 들어감.

* Software attack surface

프로그램 자체에 있는 취약점 (잘못된 프로그래밍)

* Human attack surface

사람을 공격할 수 있음 (social engineering)

ex) 주요 시스템을 다루는 직원을 공격

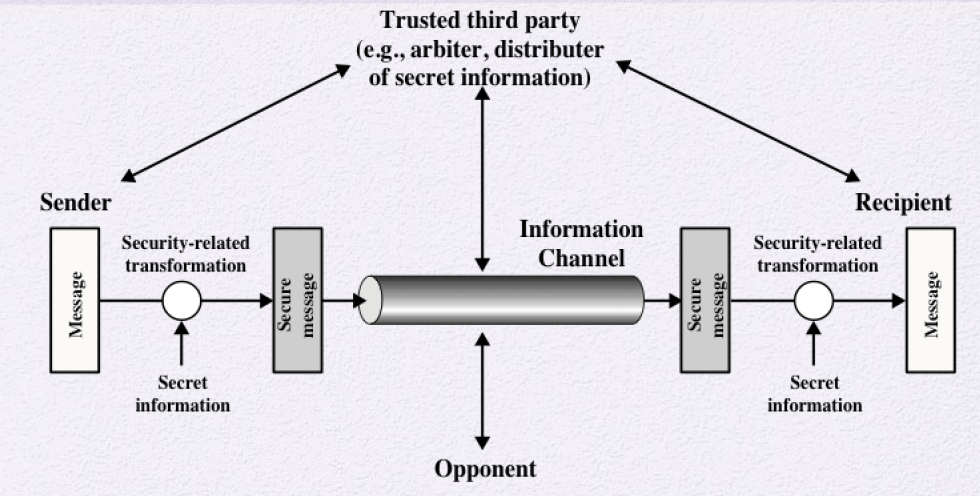
**Attack tree:** 어떤 대상을 공격하는 체계를 정리해 둔 것

* 보안상의 취약점을 이용할 수 있는 잠재적인 공격의 집합을 표현한 branching이 있는, 계층적인 data structure

**Cyber kill chain:** (최근에 많이 쓰이는 개념, 책에는 안 나와있음)

실제로 공격에 도달하기 위해서 절차상으로 공격자가 어떠한 절차를 거치는지 순서상으로 쭉 나열

**Model for Network Security**



Information channel(정보 채널)을 통해서 정보를 보내고 받는다.

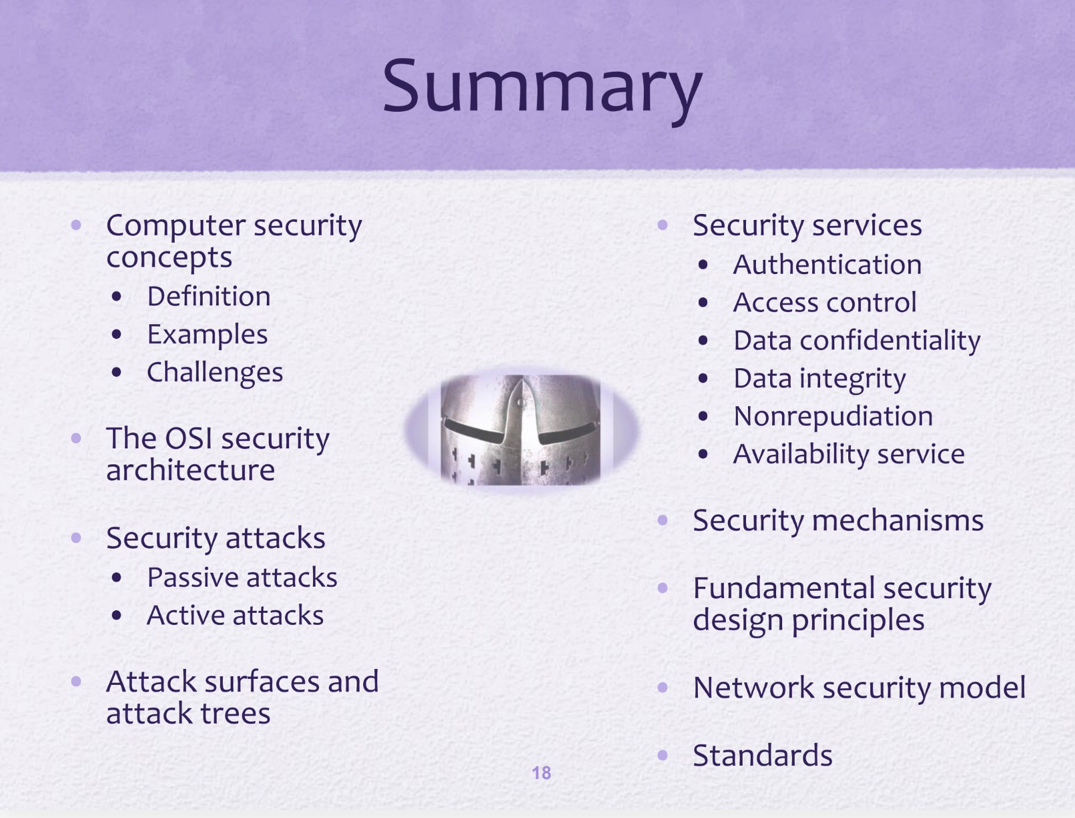
Information channel을 공격자가 항상 들여다볼 수 있는 열린 채널이라고 가정.

* 정보를 보호하기 위해서, security-related information (암호화)을 통해 secure message로 변환. 이 과정에서 비밀 정보 (키 정보)를 사용함: confidentiality 보장
* 받을 때도 똑같이 secret information을 이용해서 반대방향의 transformation을 통해 정보 받는다.
* Third party의 역할: 공격자가 중간에서 조작/몰래 보는 것을 막기 위한 절차들 필요 (security-related transformation), 그 절차를 거치기 위해 필요한 부가정보들 (secret information)을 안전하게 사용하도록 도와준다.

**Standards (표준)**

제품, 서비스, 프로토콜을 만들 때, 누가 설계하든 누가 만들든 공통적으로 지켜야할 약속

기관들

* NIST
* Internet society
* ITU-T
* ISO

(보조 자료)

**Access Control and User Authentication**

**Access Control (접근 제어)**

* 어떤 공격에 방어하는 가장 효과적인 방법
  + 공격이 일어나기 전에 미리 방지
  + 누가 어떤 정보들에 대해서 어떤 엑세스 권한들이 있는지, 그리고 어떤 것에 대해서 어떻게 허용할지 말지를 엄밀하게 정하는 것. (confidentiality, integrity)
  + Access control 이란

인가되지 않은 공개, 인가되지 않은 또는 부적절한 변경으로부터 data와 resource를 보호하는 절차

**Access Control Policies and Mechanisms**

**Access Control Policies**

* **Discretionary access control (DAC: 임의적 접근 제어)**

필수적인 접근이 아니라 사용자가 임의로 하는 것

* + 정보의 소유자가 누가 그것을 access (읽고 쓰고 실행)하는지 정한다. (privacy와 관계되는 개념)
  + 구현된 사례

Access control list(ACL): 파일에 대해서 이 파일의 소유자, 그룹, 제삼자가 어떤 작업을 할 수 있는지 정한다. 사용자의 id 기반의 access control.

* + 사용자는 주체의 id에 의해서 제약조건을 정한다.

전통적으로, file system permissions 와 DAC은 같은 개념처럼 쓰임.

DAC은 정책에 관한 것, 그 것을 구현 한 방법 중 하나가 ACL.

* + Benefit

각 주체 별로 무엇을 할 수 있고 없는지 세세하게 정할 수 있기 때문에, fine-grained control 가능

* + Disadvantage

사용자를 믿을 수 없다. 오브젝트에 대한 access control을 사용자에게 맡겨놨기 때문에, 잘못 사용하다 보면 시스템 전체적으로 문제가 생길 수 있음.

원치 않게 권한이 상승되는 경우가 생길 수 있음

* **Mandatory access control (MAC: 강제적 접근 제어)**

(MAC 2가지임: Mandatory access control, message authentication code)

* + 데이터의 주인이 아니라, 시스템 전체적으로 접근 권한을 정한다.
  + 관리자가 누가 어떻게 어떤 오브젝트를 어떻게 접근할지 정한다.
  + Rule-based access control: 규칙을 갖고 있음

주체와 객체에 각각 Security label (level of security) 부여 (중요도에 따른 계층), 매칭을 시켜서 엑세스 권한을 준다.

* + Ex) Bell-LaPadula model (전통적으로, MAC와 비슷한 말처럼 쓰임)

1. no read-up: 주체가 갖고 있는 접근 권한의 수준 (level of confidentiality) 보다 높은 레벨의 오브젝트에 엑세스 불가
2. no write-down: 주어진 레벨에 있는 주체가 더 낮은 접근 권한의 객체에 어떤 내용을 쓸 수 없다. (윗 레벨의 내용을 아래 레벨에 덮어쓸 수 없다.)
   * + - DAC에서는 의도치 않게 이러한 일이 발생 가능.
   * Modern implementation: sandboxing and Virtual machines

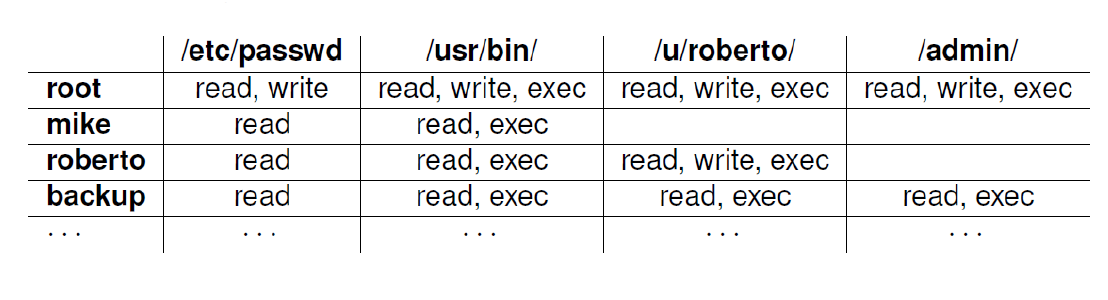
Virtual machine에서 서로 분할해서 엑세스 불가

**Access Control Models (Implementations)**

**Access Control Matrices (ACM) : 엑세스 컨트롤 행렬**

Permission을 정의하는 2차원 테이블

* 각 주체가 각 개체에 대해 어떤 동작을 할 수 있는지 권한 명세



(Table의 element) = (주체의 개수) \* (시스템에 있는 모든 파일 리소스의 객체 개수)

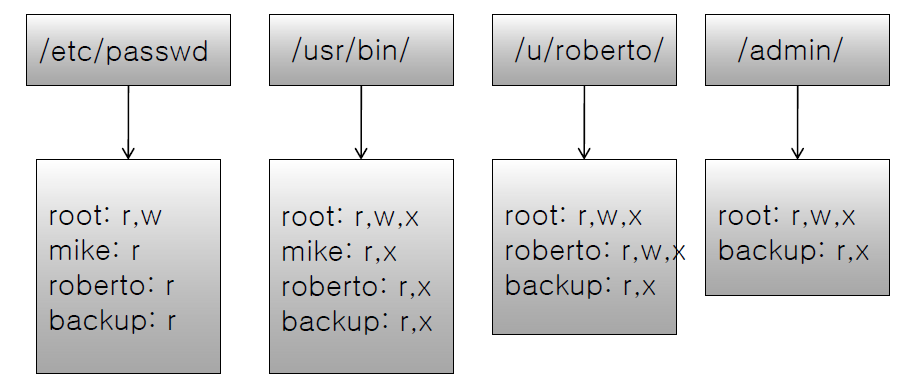
* 공간이 너무 많이 필요해서, 비효율적일 수 있음
* 보통 줄여서 사용: 비어있거나, 똑같이 구성되어있는 엔트리들 압축해서 나열

**Access Control Lists (ACL): DAC의 일종**

* 객체 위주 (접근 대상) 위주로 압축.

각 객체에 대해 주체가 어떤 권한을 갖고 있는지 나열.

아무 권한 없는 주체는 구지 써줄 필요 X



* Abbreviated ACL: UNIX file access control (간소화해서 표현)
  + Users는 Owner, group owner, all other users로 나뉜다. (id 별로 다 표시 X, 뭉뚱그려서)
  + Granularity(정밀도)는 조금 떨어짐
  + Permission은 3개로 나뉜다. (세부적인 컨트롤은 어려울 수 있음)

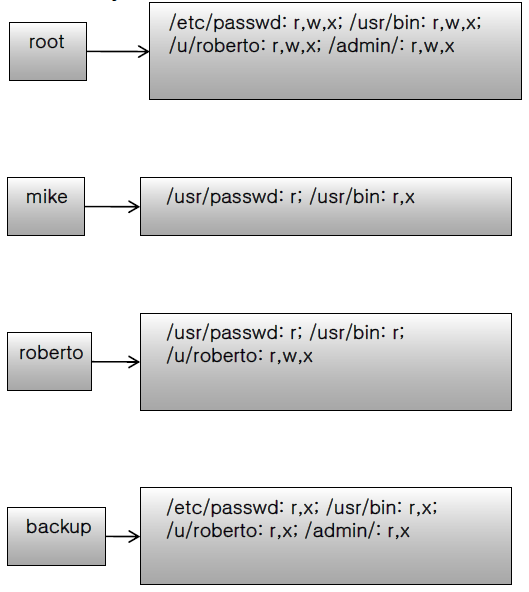
Traditional UNIX does not allow one to say, “everybody but user X.” To do this, one must create groups of all users except X.

특정부분에 대한 엑세스만을 하게하는 것 불가능

* Augmented version

Ex) recent versions of windows (권한, 성질 추가)

**Capabilities (Capability Lists: CL)**

* 주체 위주로 표시
* 일종의 티켓이라고 생각
* 별로 사용 안함
* Example:
  + 놀이공원 티켓
  + Android app permission

앱들이 시스템 자원 중에 어떤 것에 엑세스

가능한지 명시

**Role-based Access Control (RBAC): 역할 기반**

* (각 주체에 직접적으로 명시 대신,) Role을 정의하고 그 역할들에 access control 권한을 명시한다.
* Support for the principle of least privilege

(권한을 최소화 해야하는 원칙을 잘 구현한 사례)

* + RBAC-based system이 아니라면, (예를 들어 access control list 기반이라면,) file에 엑세스를 주거나 안주거나만 결정. 그 파일을 어떤식으로 엑세스해서 어떻게 변화시킬 수 있는지는 정확히 명시할 수 없다.
  + RBAC-based system에서는 그것을 자세히 명시 가능.

Ex) credit account를 사용한다. 혈당 수치에 대한 테스트를 한다. (task기반으로 자세히 명시)

* + 특정 작업에 대해 permission준다. 세부적인 control 방법

**User Authentication :** 주체가 누구인지 판단하는 것 (엑세스 컨트롤을 하기 위해서 필요함)

* Identity 또는 role이 누구인지 판단.

(identification과는 다른 개념이니 헷갈리지 말 것)

* + Identification: 여러 주체들이 섞여있는 가운데서 식별하는 것
  + Authentication: 주체가 누구라고 주장할 때, 맞는지 확인
* Authentication mechanisms
  + 사용자가 알고 있는 방법으로 인증
    - Direct input: Password, PIN(personal identification number),,,
    - Challenge-response: ex)어떤 질문을 던져 둔 뒤, password와 질문과의 계산을 통해 답변 (사람이 하기에는 무리가 있음)
    - Problem:
      * password space (PIN이 4자리라면 경우의 수가 작다- 무작위로 추측가능)
      * distribution (사람들이 자주 사용하는 password의 distribution 존재)
      * shoulder surfing (어깨 넘어로 엿보기, 소형 장치로 녹화)
      * keylogging (시스템에 침투해서 사용자가 입력하는 순간 키로깅)
    - 위험하지만 편리하고, 유출이 되면 비번 바꾸면 됨.
  + 사용자가 갖고있는 물건으로 인증
    - ID card, security token, SIM(subscriber identity module) card, RFID tag, NFC tag, OTP generator device, e-Passport, etc
    - Problem: theft or loss
  + 사용자가 하는 행위로 인증 (biometric verification 생체 인증)
    - Fingerprint, iris(홍채), face, vein(정맥),signature 등
    - Problem:
      * privacy 문제 ex) 사람의 얼굴 찍어야 됨.
      * 문제가 생겼을 때 바꾸기 어려움
    - 편리함
  + Two-factor authentication: 두가지 방법 섞어서