# 인증과 인가

컴퓨터시스템보안
금오공과대학교 컴퓨터공학
최태영

# 목차

- 인증
  - 패스워드
  - 스마트카드
  - 생체인식
- 인가
  - 접근제어
  - 다단계 보안모델
  - 은닉통로
  - 추론제어
  - 방화벽
  - 침입탐지
- OpenSSL을 이용한 침입탐지

# 인증 상황과 인증 방법의 분류

- 상황
  - 사용자는 서비스를 받기 위해서 서버에 접속을 시도함
  - 사용자와 서버는 물리적으로 가까운 위치에 있음
- 인증 방법의 분류
  - 알고 있는 것
    - 예) 패스워드
  - 소유하고 있는 것
    - 예) 열쇠, 신분증, 스마트카드
  - 본인 자신의 어떤 것
    - 예) 지문인식, 홍채 인식

# 패스워드 (Password)

- 해당 사용자만이 아는 것으로 가정하는 정보
- (부분적으로) 패스워드가 될 수 있는 정보
  - 생년월일, 주민등록번호, 출생지, 전화번호
  - 쉽게 노출될 수 있으므로 안전하지 않다.
- 공격자들의 공격 패턴
  - 대상과 관련 있는 단어들을 시도
  - 관련 단어들의 조합을 시도
  - 일반적인 단어들을 시도
  - 일반적인 단어의 조합을 시도
  - 전수조사 또는 공격의 대상을 바꿈
- 난수 (random number)
  - 공격 패턴의 마지막에 대상이 되는 값
- 패스워드는 대칭키 암호화의 키와 유사한 성질을 가지게 됨
  - 공격자가 보다 많은 전수조사를 하게끔 충분한 길이를 가져야 함
  - 길이가 긴 난수와 같은 패스워드는 기억하기 어려움

# Passphrase

- 단어 대신 문장을 이용하여 인증하는 방식
  - id : alice
  - pw : boating with Tom was very exciting
- 문장은 경우의 수를 매우 증가시킴
- Linux, OpenBSD, Solaris, FreeBSD, NetBSD, OpenSSL 등에서 지원
- Password만 지원되는 경우에 passphrase 효과를 얻기
  - 문장의 두 문자들을 조합함
    - 문장 : The day April 6 when I met Dorothy
    - Pw : TdA6wImD
  - 한글 문장의 경우
    - 문장 : 어젯밤에 먹은 군고구마 4개
    - 선택한 자음 또는 모음 : ㅇ ㅈ ㅂ ㅇ ㅔ ㅁ ㄱ 고 4 ㄱ
    - Pw : d w q d p a r r h 4 r

# Password Management

- 패스워드 관리의 주체
  - 일반적으로는 사용자
  - 시스템 관리자가 관리하는 경우 : 짧은 기간 사용
- 관리자의 역할
  - 주기적으로 사용자들의 패스워드를 검사해야 함 → 패스워드 크랙 프로그램을 사용할 수 있음
    - 하나의 허술한 패스워드는 시스템 전체에 위협이 됨
    - 허술한 계정으로 침입 뒤 패킷이나 패스워드 파일 등을 검사하여 관리자 계정으로 침입 가능
- 패스워드의 변경
  - 주기적으로 변경해야 함
  - 사용자는 패스워드 변경을 싫어함
    - 기억하기 쉬운 패스워드로 바꾸거나
    - 몇 개의 패스워드들을 번갈아 가며 사용

# Password MGNT (Cont.)

- 너무 많은 패스워드들
  - 인터넷의 발달로 인한 다양한 서버 존재
  - 사용자들은 각 서버마다 다른 패스워드를 기억하기를 싫어함 → 동일한 패스워드 사용
  - 하나의 취약한 서버는 전체 서버의 안전성을 떨어뜨림
    - 패스워드가 인증을 위한 용도로서는 안전하다고 할 수 없음
- Online attack
  - 공격자는 log-in 창에 대상의 id와 password를 추측하여 입력
  - 추측이 맞을 때까지 password를 바꿔가며 공격을 계속함
  - 공격 판단 : 연속적인 패스워드 실패와 시도가 되풀이 되는 경우
  - 대응책 : 계정 잠금

# 계정 잠금

- Online attack으로 판단될 때 계정 접속을 더 이상 허용하지 않는 방법
- 예) ATM에서 3번 연속으로 비밀번호 입력 실패 시 계정을 잠금
- 잠긴 계정을 풀기 위해서는 관리자에게 요청해야 함
- 서비스 거부 공격을 발생시킬 수 있음
  - 공격자가 고의로 각 계정에 틀린 패스워드들을 연속해서 입력함
  - 관리자 계정은 같은 이유로 잠금을 할 수 없음
- 대응방법 : 다단계 계정
  - 계정에 접속하는 첫 번째 단계에서는 계정 잠금을 하지 않음
  - 중요한 내부 서비스를 위한 두 번째 단계에서는 계정 잠금을 함
  - 사용되는 password는 각각 달라야 함

# 패스워드 저장

- 파일의 형태로 컴퓨터시스템의 저장장치에 그대로 저장하는 방법
  - 공격 : Online attack을 통해 시스템에 들어온 공격자가 패스워드를 모두 얻을 수 있음
- 패스워드의 해시들을 파일에 저장하는 방법
  - 로그인 과정 시에 입력된 패스워드를 해시하여 얻은 값과 저장된 패스워드 해시를 비교
  - 공격자는 online attack으로 시스템에 들어오더라도 패스워드를 얻을 수 없음
  - 공격자는 패스워드 해시 파일을 자신의 컴퓨터에 옮겨 offline attack을 시도함

# Offline Attack

- 공격자의 상황
  - hashed password file 획득
  - 패스워드 해시 프로그램 소유
  - 패스워드 사전 소유
    - 패스워드가 될 수 있는 단어들의 모임
- 공격방법 : 패스워드를 추측한 수 해시를 적용하여 같은 값인가 확인
- 공격대상
  - 특정 시스템의 특정 사용자
  - 특정 시스템의 임의 사용자 : 특정 시스템 공격에 유용
  - 임의 시스템의 특정 사용자
  - 임의 시스템의 임의 사용자

# 오프라인 공격의 위험성

- 상황
  - 패스워드 사전에 1백만 개의 단어
  - 패스워드가 사전에 있을 확률 : 10%
  - 패스워드 파일내의 패스워드 : 1,000 ($\approx 2^{10}$)개
  - 가능한 패스워드 개수 : $2^{64}$개
- 공격
  - 1개의 패스워드를 추측하고 그 해시를 1,000개의 파일 내의 해시와 비교
    - 일치하는 것이 있으면 공격 성공
  - 패스워드 사전을 사용하지 않을 때
    - 평균 $2^{64}/(2 \times 1000) \approx 2^{64}/2^{11} = 2^{55}$ ← 공격 곤란
  - 패스워드 사전을 사용할 때
    - 최소한 1개의 패스워드가 사전에 있을 확률 : $1 - 0.9^{1000} \approx 1$
    - 평균 해시 회수 : $2^{20-1}/2^{10} = 2^{9}$ : 약 500회의 해시 작업으로 패스워드 추측 성공

# Salting

- Salt
  - 각 password마다 주어지는 random number
  - 사용자마다 다른 값을 가짐
  - 암호화되지 않은 채로 password file에 저장됨
- Password file에는 다음 정보들이 저장됨
  - User ID
  - h(pw | s)
  - Salt s
- 공격자의 해시 연산 회수 증가가 목적
  - 각 해시는 각기 다른 salt가 포함되어 있다.
  - 한 사용자의 password를 공격하려면 지정된 salt를 사용해야 함

# Salting (Cont.)

- 한 사용자 공격을 위해 패스워드 사전의 패스워드들의 해시를 구하는 회수
  - 동일한 해시가 발견될 경우 : 평균 500,000
  - 해시가 없을 경우 : 1,000,000
  - 10%의 해시가 있을 확률을 적용할 때 한 사용자 공격을 위한 해시 계산 회수
    - $0.1 \times 2^{19} + 0.9 \times 2^{20} \approx 9.96 \times 10^5$
  - 첫 번째 사용자 공격 실패 후 두 번째 사용자 공격 시도
    - $0.1 \times 2^{19} + 0.9 \times 0.1 \times (2^{20} + 2^{19}) + 0.9^2 \times (2 \times 2^{19} + 2^{20})$
  - 1000개의 패스워드에 대해서

    $0.1 \times 2^{19} +$

    $0.9 \times 0.1 \times (2^{20} + 2^{19}) +$

    $0.9^2 \times 0.1 \times (2 \times 2^{20} + 2^{19}) +$

    $0.9^3 \times 0.1 \times (3 \times 2^{20} + 2^{19}) +$

    $\cdots +$

    $0.9^{999} \times 0.1 \times (999 \times 2^{20} + 2^{19}) \approx 9.961 \times 10^6 < 2^{24}$

# Smart Card


From Wikipedia

- Processor, memory, 그리고 cryptography module 등이 내장된 명함 크기의 카드
- Public key certificate, private key 등을 저장할 수 있음
- Smart card reader가 CRL을 쉽게 얻을 수 있는 경우 다음 과정으로 인증이 이루어짐
  1. 난수값 (nonce) : 시스템 → 스마트카드
  2. 공개키 인증서 : 스마트카드 → 시스템
  3. 난수값에 대해 개인키로 작성한 전자 서명을 첨부 : 스마트카드 → 시스템
  4. 시스템은 전자 서명을 검증하고 CRL 목록에 공개키 인증서가 없는가를 확인
- 스마트 카드의 현 소유자가 스마트 카드의 주인인지 확인해야 함
  - 사용자에게 PIN 입력 요구
  - 공격자의 비밀번호 추측을 막기 위해 카드 잠금을 하기도 함

# 생체인식

- 사용자 자신으로부터 신원을 확인
  - 얼굴 인식, 자필 서명 등 이미 사용되는 방법
- 장점 : 기억하거나 소지할 필요가 없음
- 단점 : 인식 장치가 명확히 구분하기가 어려움
  - 사람의 신체는 끊임없이 변화함
  - 주위 환경의 변화에도 민감함
  - 다른 사람이라도 유사한 특성이 있음
- 인식 과정
  - 데이터 등록 단계 : 시간이 걸리고 비용이 고가임
  - 인식 단계
- 식별에도 사용될 수 있음
  - 식별 : 여러 사람들 중 특정 대상을 찾는 것
  - 여러 대상으로부터 가장 유사한 대상을 찾아야 하므로 인증보다 복잡함
  - 대상의 비협조 문제도 발생함

# 생체 인식상의 오류

- 기만율 (flaud rate)
  - 다른 사람이 자신으로 인증되는 비율
- 모욕률 (insult rate)
  - 사용자가 인증 받지 못하는 비율
- 생체 인식에서의 판정은 인식한 수치에 대해 threshold값을 적용시키는 과정
  - 한 쪽의 비율이 높아지면 다른 쪽 비율이 낮아짐
- 동일 오류율 (equal error rate, EER)
  - 스레시홀드를 조절하여 기만율과 모욕률을 동일하게 만들었을 때의 오류율
  - 생체 인식 시스템의 성능을 비교할 때 주로 사용됨

# 생체 인식 방법

- 지문인식 (Fingerprint recognition)
  - 지문은 사람마다 다르며 시간이 경과해도 좀 처럼 바뀌지 않음
  - 지문의 형태 : 제상문, 와상문, 궁상문
  - 특징점 : 지문의 선이 모이거나 갈라지는 점
- 장형인식 (Palm print recognition)
  - 손과 손가락의 폭과 길이 등과 같은 기하학적 형태 분석
  - 각 손가락 마디의 폭, 손가락들의 길이, 손바닥의 너비 등을 데이터로 저장
  - 인식 과정이 빠름
  - 손의 형태가 빨리 바뀌고 유사한 경우가 많음 (EER 증가)

# 생체 인식 (계속)

- 홍채인식 (Iris Scan)
  - 홍채를 이미지로 현상한 뒤 특성 저장
  - 유전적 특성이 거의 없고 일생 동안 변하지 않음
  - 데이터 처리
    - 홍채를 흑백사진 촬영 후 2차원 wavelet transform 적용 → 256 byte iris code 저장
  - 코드 비교
    - Hamming distance : 일치하지 않는 bit수의 비율
      - 일치하지 않는 bit가 k개 : k/2048
    - 일반적인 threshold : 0.32 미만이면 일치
  - 단점 : 실제 눈이 아닌 사진으로 위장할 수 있음
    - 대비책 : 밝은 빛으로 동공 수축을 확인 후 인식

# Authentication

- 사용자가 인증을 받아 시스템 접근을 허락 받은 뒤 어떤 기능 또는 서비스를 이용하도록 할지를 다루는 부분
  - 예) 앨리스는 서버에 일반 사용자로 인증을 받아 로그인을 한 상태이면 관리자 기능을 수행할 수는 없다.
- Access control (접근 제어)
  - 주체가 객체에 어떤 기능을 요청할 때 허락할지 여부를 판단하는 것
  - 허락 여부는 주체와 객체간의 권한 여부에 의해 결정됨
    - 특정한 자료구조에 저장됨
- 자료구조
  - 접근제어 행렬
  - 접근제어 목록

# Access Matrix

- 또는 access control matrix, 접근제어행렬
- 사용자가 각 행을 대표하고, 자원 또는 서비스가 각 열을 대표
- 행렬의 각 칸 M(i,j)는 사용자 i가 자원 j에 접근할 권한을 표시
  - 예) M(4, 3) = 읽기/실행 : 사용자 4는 자원 3에 대해서 읽기와 실행하기 연산을 수행할 수 있음
- 접근제어행렬은 빠른 시간에 권한 확인이 가능
- 과도한 기억공간 소모
  - 사용자 100명, 프로그램 500개, 파일을 포함한 자원이 2만개이면 12백만 개의 방으로 이루어진 행렬이 필요

| | 파일 A | 디렉토리 B | 컴파일러 |
|---|---|---|---|
| 앨리스 | 읽기/쓰기 | 읽기/쓰기 | 실행 |
| 밥 | 읽기 | 읽기 | |
| 컴파일러 | 읽기 | | 실행 |

# Access Control List

- ACL, 접근제어 목록
- 각 자원에 대해 접근이 허락된 주체들의 권한을 담은 리스트
  - 예) 디렉터리 B : (앨리스, 읽기/쓰기), (밥, 읽기)
- 권한 목록 (Authrization list)
  - 각 주체에 대해서 접근할 수 있는 자원들과 연산의 리스트
  - 예) 밥 : (파일 A, 읽기), (디렉터리 B, 읽기)

# Multilevel Security

- 다단계 보안모델 (MLS)
- 정보 또는 객체들을 중요도에 따라 여러 단계로 나눔
- 각 주체들이 각 객체에 대한 접근 허용 범위에 대한 규정
- 예) 미 국방성의 문서에 대한 security level
  - top secret
  - secret
  - confidential
  - unclassified
- 취급자에 대해서 취급 등급이 주어짐
- Bell-LaPadula model
- Biba (integrity) model

# Bell-LaPadula (BLP) Model

- 가장 기본적인 형태의 다단계 모델
- 1970년 초에 David Elliott Bell과 Leonard J. La Padula에 의해 제안
- 2개의 mandatory access control (MAC)과 1개의 discretionary access control (DAC)로 구성
  - The simple security property : 주체 S는 자신보다 높은 등급의 객체 O를 읽을 수 없다.
  - *-property : 주체 S는 자신보다 낮은 등급의 객체 O에 쓸 수 없다.
  - 자유재량 (discretionary) 보안 특성 : 접근제어를 위해 접근제어 행렬을 사용한다.
- 정보는 보안 정도가 낮은 곳에서 높은 곳으로 흐르도록 규정함
- 너무 엄격한 제약을 두는 모델
  - 문서의 유효기간을 어떻게 반영할 것인가?

# Biba Model

- 다단계 모델의 무결성 규칙
- 1977년 Kenneth J. Biba가 제안
- 보안 규칙
  - The simple integrity axiom : 주체 S는 자신보다 낮은 무결성 수준을 가진 객체 O를 읽을 수 없다.
  - The *-integrity axiom : 주체 S는 자신보다 높은 무결성 수준을 가진 객체 O에 쓸 수 없다.
- 의미 : 자신보다 낮은 수준의 오염된 정보를 읽어선 안되고 자신보다 높은 수준의 문서를 오염된 정보로 손상시켜서는 안 된다.
- 예제
  - 군대에서의 명령 전달 체계 : 명령은 하부에서 상부로 지시되지 않는다.
  - 공신력 있는 기관은 검증되지 않은 정보를 발표하지 않는다.

# 다측면 보안

- 다양한 정보들을 하나의 계층 내에 포함시키기는 곤란함
  - 예) 기존 시스템은 의료 시스템, 새로 추가된 요소는 AIDS 관련 분야
    - 일반적인 의료 정보에 접근할 수 있는 의사가 AIDS 관련 내용을 허가 없이 접근하는 것은 위험함
- 다단계 보안 모델에 구역이라는 개념이 추가된 모델
- 각 구역마다 다단계 보안 모델이 적용됨
  - 한 구역에서 주어진 주체의 권한이 다른 구역에서 적용되지는 않음
- 적용 예
  - 1급{의료}, 2급{복지}, 2급{의료, 복지}
  - 1급{의료} 등급 취급자는 2급{복지} 등급 객체에 접근할 수 없음

# Covert Channel

- 은닉통로는 시스템 설계자가 고려하지 않았던 통신 채널을 말함
- 내부공격자가 정보를 유출하기 위해서 찾아낸 접근권한이 정의되지 않은 경로
- 예제
  - UNIX의 /tmp directory내의 file
    - 다른 사용자가 그 파일의 내용을 읽을 수는 없지만 존재 유무를 알 수 있음
  - CPU 사용률의 상태
    - CPU 사용률이 높은 경우와 낮은 경우를 각각 정보로 구분함
  - TCP 패킷의 헤더의 예비 필드, 일련 번호 필드

# 파일 생성 및 삭제를 이용한 은닉통로

| 시각 | 내부 공격자 | 외부 공격자 | 내용 |
|---|---|---|---|
| 1초 | 파일 생성 | | |
| 1.5초 | | 파일 체크 | 1 |
| 2초 | | | |
| 2.5초 | | 파일 체크 | 1 |
| 3초 | 파일 제거 | | |
| 3.5초 | | 파일 체크 | 0 |
| 4초 | | | |
| 4.5초 | | 파일 체크 | 0 |
| 5초 | 파일 생성 | | |

# Covert Channel

- 은닉 통로의 조건 (동시 만족 필요)
  - 내부 공격자와 외부 공격자가 어떤 형태로든 접근할 수 있는 공유자원이 있다.
    - 인터넷 등으로 연결되어 있으면 항상 공유자원이 존재한다.
  - 내부 공격자는 그 공유자원의 특성 또는 존재 여부 등을 변경할 수 있다.
    - 일반적으로 이러한 자격을 가진 내부 공격자가 은닉 통로를 이용한다.
  - 내부 공격자와 외부 공격자는 서로 간에 동기화를 할 수 있다.
    - 컴퓨터 프로그램을 통해서 쉽게 동기화를 할 수 있다.
- 위 조건들 중 하나라도 만족 못하게 하는 것은 어렵다.
- 은닉 통로의 throughput을 줄일 수는 있음
  - 예) CPU 사용률을 이용한 은닉통로를 막기 위해 랜덤한 CPU 과부하를 발생
- 은닉 통로의 대역폭을 줄이는 행동은 시스템의 성능을 저하시키는 원인이 됨
  - 예) 과부하 발생 프로그램은 다른 정상적 프로그램의 수행을 지연시킴

# Inference Control

- DB에 저장된 정보는 중요도에 따라 공개되는 정보와 그렇지 않은 정보로 나뉜다.
- Inference attack : 공개되는 정보들만을 바탕으로 공개되지 않은 정보를 유추하는 행위
  - 예1 : 결석자가 있는 교실과 특정 질병이 특정 학교에서 발생했다는 정보로 누가 특정 질병을 가졌는지 알아내는 행위
  - 예2 : 이름이 A로 공개된 연예인 사건에 대해 관련 정보를 조합하여 A의 본명을 찾아내는 행위
- 추론제어 : Inference attack이 발생하지 않도록 공개되는 정보의 정도를 조절하는 행위
  - 질의에 대한 결과가 특정한 크기 이하이면 공개를 제한
    - 특이한 현상을 연구하는 경우에 자료 수집을 방해
  - 자료에 잡음을 부여하여 일부러 정확하지 않은 자료를 공개
- 추론 제어는 민감한 정보와 정확한 결과 제공 사이의 딜레마

# Firewall

- 방화벽 : 허락 받지 않은 패킷들을 차단함으로써 외부 공격으로부터 subnet을 보호하기 위한 컴퓨터 시스템
- 정해진 규칙에 따라 통과 가능한 패킷과 그렇지 못한 패킷을 규정함
- 동작 범위 : 개인 컴퓨터 ~ LAN 단위의 네트워크
- 위치 : 주로 내부 인터넷과 외부 인터넷을 연결하는 gateway에 설치
- 성격에 따라
  - packet filter, application gateway, proxy server 등이 있음

# Packet Filter

- 기초적인 형태의 firewall
- Network layer에 해당하는 TCP / UDP packet을 검사하여 통과 여부를 결정
- 검사 항목
  - source / target IP address
  - port number, TCP flag
- 조건 설정 : Access Control List 이용
  - 허용되는 또는 허용되지 않는 주소 또는 포트 번호 입력
- 단순한 매칭을 수행하므로 속도가 빠름
- 규칙을 우회하는 공격에 약함
  - 예) 바이러스가 포함된 허용된 ip로부터의 메일

# Cisco System Router에서의 ACL 관리 예

```
access-list 1 permit tcp 202.31.100.0 0.0.0.255

access-list 1 permit 128.100.211.0 0.0.0.255

access-list 2 permit udp 200.100.0.0 0.0.255.255

int s 0
> ip access-group 1 in
> ip access-group 2 out
> ^Z
```

# Application Gateway

- 응용 게이트웨이, 응용 프록시 (application proxy) 등으로도 불림
- 일반적으로 방화벽에서 수행되는 응용프로그램
- Client는 application gateway에게 server에 접속하고 서비스 수신을 대신 하도록 부탁함
  - Client가 server에게 보이지 않도록 함
- Application gateway는 수신된 서비스에 대해 바이러스나 오류를 체크한 뒤 client에게 전달
  - 예) email 서비스의 경우 spam mail 삭제를 수행할 수도 있음
- 장점 : 다양한 수준의 필터링을 수행할 수 있음
- 단점 :
  - 패킷을 응용프로그램의 데이터 단위로 결합해야 하므로 오버헤드 발생
  - 암호화된 통신을 처리하기에 곤란함

# Proxy Server

- 부분적으로 server 역할을 함으로써 client가 server에 접속하지 않고도 server가 제공하는 서비스를 이용하도록 하는 server
- Proxy server와 application gateway의 차이
  - Proxy server는 반드시 gateway에 존재할 필요 없음
  - Proxy server는 caching을 수행함
- 다양한 proxy server
  - Caching proxy server, web proxy, anonymizing proxy server, hostile proxy, intercepting proxy server, transparent proxy server, reverse proxy server, circumventor, content filter, etc.

# Proxy Servers

- Caching proxy server
  - 이전에 받은 데이터를 caching해 두었다가 요구하는 client에게 전달
  - 네트워크 성능 향상에 도움
- Web proxy
  - 주 기능은 웹 페이지 캐싱이지만 필터링 기능도 수행한다. (content filtering web proxy)
  - Browser에 맞는 양식의 web page로 변경하는 작업도 수행함 (portable browser 경우)
- Anonymizing proxy server
  - Client가 자신의 ip 정보를 server에게 드러내지 않도록 해주는 서버 (헤더 정보를 변경함으로써 구현됨)
  - 스팸 메일의 온상이 되어왔음

# Proxy Servers (Cont.)

- Hostile proxy
  - 공격자가 악의를 가지고 희생자가 사용하는 게이트웨이나 컴퓨터에 설치하여 통신을 엿듣는 프로그램
- Intercepting proxy server
  - Gateway와 proxy server를 결합한 형태
  - 내부망의 내외로 이동하는 모든 통신을 제어할 수 있음
- Transparent proxy server
  - 사용자가 proxy의 존재를 알지도 못하고 굳이 알 필요도 없도록 만들어진 proxy
  - 반대 개념 : non-transparent proxy server
- Reverse proxy server (next page)

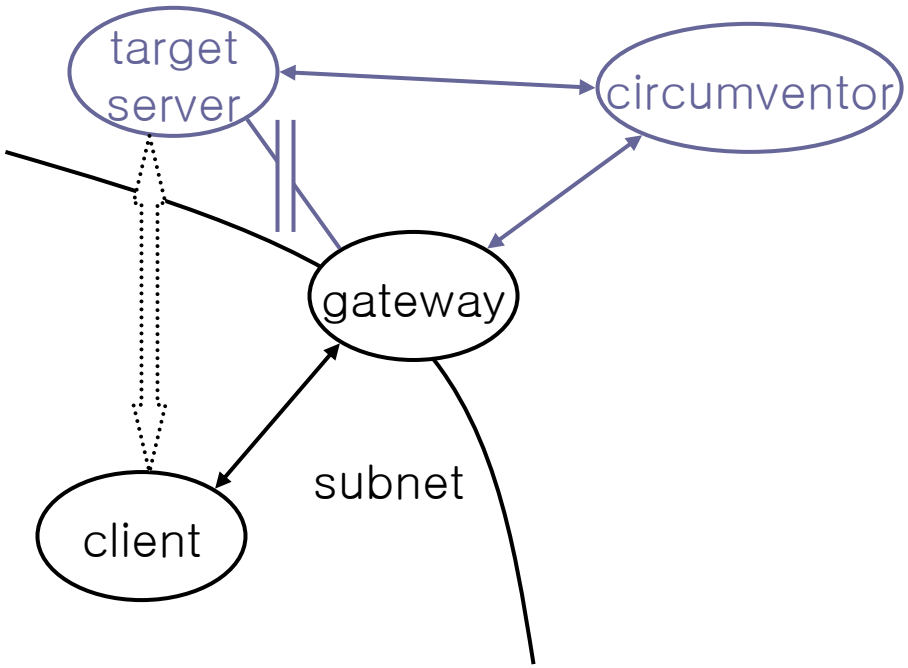
# Reverse Proxy Server

- 1개 이상의 server들로 구성된 server group과 인접하여 그 server로 가는 모든 요구가 자신을 경유하도록 하는 proxy
- Server 작업의 효율화를 지원
  - cryptography : SSL hardware accelerator등을 이용하여 server의 암호화 부하를 줄임
  - load balancing : 비교적 부하가 적은 server에 작업을 할당함으로써 전체 server의 부하를 비슷하게 유지
  - caching : 자주 언급된 정보들 보관해두고, 그 요청이 발생하면 server에 보내지 않고 직접 처리
  - compress : network throughput이 낮은 client 대상으로 효과적
  - security : proxy가 server로 향하는 공격을 대신 받음

# Proxy Servers

- Circumventor
  - proxy server를 이용하여 blocked site를 우회 접근할 수 있도록 하는 proxy
  - firewall은 특정 site를 ip address로 기입하여 차단함 (black list)
  - circumventor를 firewall의 black list에 포함되지 않은 것으로 선택
  - 사용자는 circumventor에게 대상 server에 대한 요청을 대신 수행하도록 하고 그 결과를 받음
- Content filter
  - 동기 : 단순한 packet filter는 circumventor와 같은 우회 방법을 막지 못함
  - 행동 : 질의 패턴, 패킷의 내용을 분석하여 차단 여부를 결정
    - 각 내용을 스포츠, 포르노, 온라인 쇼핑, 도박, 게임, 뉴스, 시사 등으로 구분
    - 키워드를 검색하여 구분
    - 바이러스 패턴을 검색하여 바이러스를 필터링 하기도 함

# Circumventor



# Intrusion Detection

- Intrusion : 자원에 대해 허락 받지 않은 작업을 시도하는 행위
- Intrusion detection : intrusion이 발생했거나 발생하고 있음을 알아차리는 작업
- Intrusion detection system (IDS) : intrusion detection을 수행하기 위해 개발된 system
- IDS의 침입 탐지 방식 : 비 정상적인 활동 판단
  - 예) 10회 이상 패스워드 시도
  - 예) 평소에 접근하지 않는 파일에 대한 지속적 접근 시도
- IDS의 분류 (탐지 방법에 따라)
  - Signature-based IDS
  - Anomaly-based IDS
- 탐지 대상에 따른 IDS 분류
  - Host-based IDS
  - Network-based IDS (NIDS)

# Signature-based IDS

- 특정 서비스 신청 등에 대한 기록을 바탕으로 침입 여부를 판단
- 기존 공격의 특성을 DB에 저장해 두고 event (packet 도착, file download 등)가 발생할 때마다 검사
- 예제
  - 연속적인 로그인 시도
  - 컴퓨터 바이러스 백신 프로그램의 패턴 (virus signature) 비교
  - Network packet을 검사하여 공격 packet인지를 판단
- 장점 : 알려진 형태의 침입에 대해서는 매우 빠르고 정확하게 탐지
- 단점
  - 새로운 형태의 침입은 탐지하지 못함
  - 공격형태가 많아짐에 따라 pattern matching overhead 증가

# Anomaly-based IDS

- 통계적 방법으로 정상 상태를 규정한 뒤, 현재 상태에 대한 통계치가 정상 상태와 다르다고 판단되면 경고를 생성하는 시스템
- 관련 방법론 : statistic approach, feature selection, Bayesian statistics, covariance matrices, belief network, predictive pattern generation, neural networks, etc.

# Statistic Approach

- 요소 $i$, $(1 \leq i \leq n)$
- 정상상태 사용률 $M_i$
- 검사상태 사용률 $S_i$
- Geometric distance

$$R = (M_1 - S_1)^2 + (M_2 - S_2)^2 + \cdots + (M_n - S_n)^2$$

- 비정상 상태 판단
  - 미리 설정한 임계치 $H$에 대해 $R > H$ 인 경우

# Statistic Approach (Example)

| 요소 | 내용 | 최대치(1주일간) | 정상($M_i$) | 측정치($S_i$) |
|---|---|---|---|---|
| 1 | 계정 접속 빈도 | 12회 접속 | 0.4 | 0.5 |
| 2 | File A read freq. | 70회 읽기 | 0.3 | 0.2 |
| 3 | File B write freq. | 14회 쓰기 | 0.7 | 0.8 |
| 4 | Server C access freq. | 4회 접속 | 0.2 | 0.35 |
| 5 | File D execute freq. | 50회 실행 | 0.45 | 0.41 |

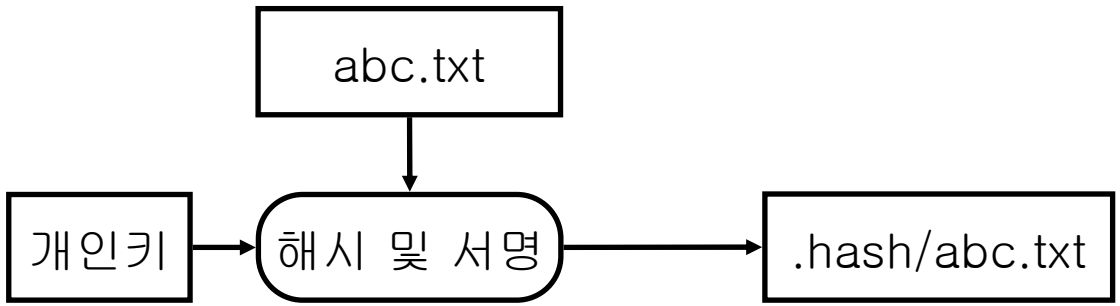
$$R = (0.4-0.5)^2 + (0.3-0.2)^2 + (0.7-0.8)^2 + (0.2-0.35)^2 + (0.45-0.41)^2 = 0.0541$$

$$R = 0.0541 < 0.1 = H$$

- 임계치 H를 0.1이라 할 때 → 정상상태 판정

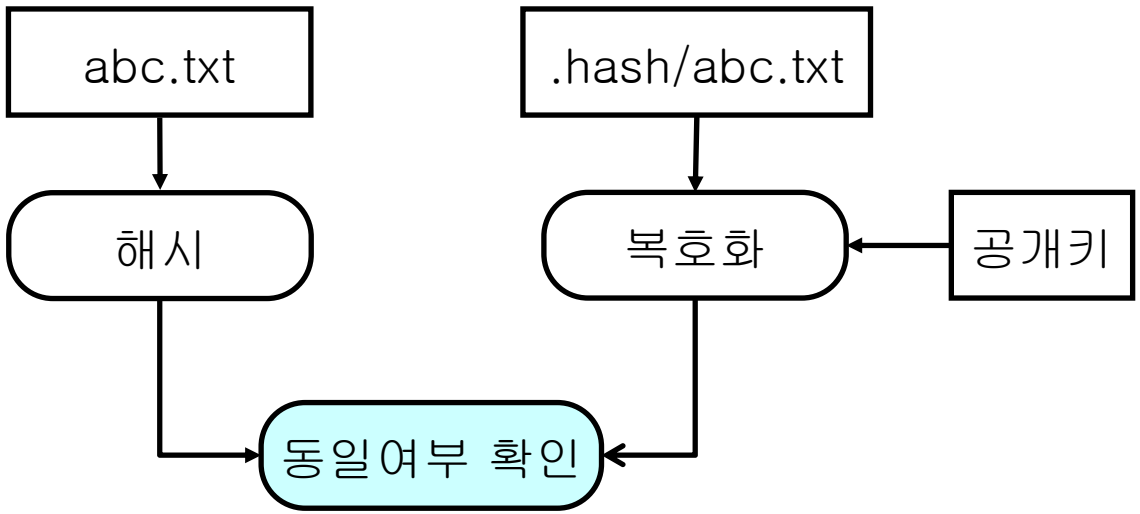
# OpenSSL을 이용한 침입탐지

- 각 파일의 전자서명을 .hash subdirectory에 저장
- 각 파일을 저장된 전자서명으로 검증함



# 검증 과정

- 파일이나 전자서명 중 하나라도 변경되면 서로 일치하지 않는다는 메시지를 출력



# 컴파일 및 수행 과정

- Source file editing and compile
  - `Edit hashNsign.c`
  - `gcc -g -Wall -o hashNsign hashNsign.c verify.c -lcrypto`
- Initialize or make signatures
  - `./hashNsign -i AlicePriv.pem`
- Intrusion detection
  - `./hashNsign -c AlicePub.pem`
- Main code
  - 현재 디렉터리의 파일들에 대해 signature를 만들거나 검증을 함
  - Subdirectory에 대해서는 작업을 수행하지 않음
  - opendir(), readdir(), closedir() 함수를 이용하여 디렉터리의 모든 파일들을 순서대로 처리함
  - stat() 함수와 S_ISREG() 매크로를 통해 일반 파일인지 확인

# Main Code

```
assert(dirp = opendir("."));
while((dentry = readdir(dirp)))
  if(dentry->d_ino != 0){
    res = stat(dentry->d_name, &statBuf);
    if (res == 0 && S_ISREG(statBuf.st_mode)){
      if (strcmp(argv[1], "-i")==0)
        hashNsign(argv[2], dentry->d_name);
      else if (strcmp(argv[1], "-c")==0)
        verify(argv[2], dentry->d_name);
      else
        assert(1);
    }
  }
closedir(dirp);
```

# Structure of hashNsign(*p*, *n*)

- Parameters
  - 1번째 arg *p*: 개인키 파일명
  - 2번째 arg n: 전자서명될 파일명
- Structure
  - 개인키를 읽어 EVP 라이브러리에 적합한 자료구조로 변환
    - 개인키 파일을 읽는 PEM_read_RSAPrivateKey() 함수
    - EVP_PKEY_set1_RSA() 함수로 키 구조체 변환
  - 파일을 읽어 전자 서명을 생성
    - EVP_SignUpdate() 함수를 여러 번 수행
  - 전자 서명을 .hash 서브디렉터리에 저장
    - provideSubdir() 함수를 통해 subdirectory path 입력
      - abc.txt → ./.hash/abc.txt

# hashNsign()

```
fp = fopen(privFile, "rb");
rsaPriv = PEM_read_RSAPrivateKey(fp, NULL, NULL, NULL);
fclose(fp);
pkey = EVP_PKEY_new();
res = EVP_PKEY_set1_RSA(pkey, rsaPriv);

/* make signature – 5장의 전자 서명과 유사 */

provideSubdir(sfn, dirn, fn);
if (stat(dirn, &finfo)==-1)
  mkdir(dirn, 0700);
fp = fopen(sfn, "w");
fwrite(sign, 1, signSize, fp);
fclose(fp);
```

# verify()

```
fp = fopen(pubFile, "rb"); assert(fp);
rsaPub = PEM_read_RSAPublicKey(fp, NULL, NULL, NULL);
fclose(fp);
pkey = EVP_PKEY_new();
res = EVP_PKEY_set1_RSA(pkey, rsaPub);

provideSubdir(sfn, dirn, fn);
fp = fopen(sfn, "r"); assert(fp);
signSize = fread(sign, 1, SIGNSZ, fp);
fclose(fp);

/* verify and clear memory – 5장의 전자서명 검증과 유사 */
```

# 제 5 차 프로젝트

- 메시지를 암호화하고 서명하여 보내는 과정에서 공개키 인증서를 사용한다.
- P5a
  - 난이도 : 쉬움
  - 내용 : 공개키는 항상 공개키 인증서에 포함된 것만을 사용하는 채팅 프로그램을 구현하시오. 미리 상대방의 공개키 인증서를 가지고 있다고 가정한다.
  - 비고 : 제 4 차 프로젝트를 하지 않아도 구현할 수 있다.

# P5b

- 난이도 : 보통
- 내용 : 프로그램의 수행을 처음 시작하면 항상 자신의 공개키 인증서와 개인키가 없다고 가정하며 다음과 같은 과정을 거쳐 공개키 인증서를 얻고 유효성을 확인한 후 채팅을 시작한다.
  - 자신의 개인키와 인증서 서명 신청서를 만든다.
  - 인증서 서명 신청서를 CA에게 제출하여 자신의 공개키 인증서를 CA에 등록하고, 인증서도 받는다.
  - 상대방의 공개키 인증서를 CA에 요청해서 받는다. CA에게 해당 인증서가 없는 경우에는 어느 정도 기다린 후 다시 요청한다.
  - 자신과 상대방의 유효한 공개키 인증서를 모두 확보한 후 채팅을 시작한다.
- 비고 : 4차 프로젝트를 한 경우에만 구현할 수 있다.