Computer Security: Principles and Practice

Chapter 3 – User Authentication

User Authentication

- ➢ fundamental security building block
  - basis of access control & user accountability
- ➢ is the process of verifying an identity claimed by or for a system entity
- ➢ has two steps:
  - identification - specify identifier
  - verification - bind entity (person) and identifier
- ➢ distinct from message authentication

User Authentication

- User authentication example:
  - User real name: *Alice Toklas*
  - User ID: *ABTOKLAS*
  - Password: A.df1618hJb
- These informations are stored in a system
  - Only Alice can access with this credential
  - But attackers can still do something ...

Means of User Authentication

- four means of authenticating user's identity
- based on something the individual
  - knows - e.g. password, PIN
  - possesses - e.g. key, token, smartcard
  - is (static biometrics) - e.g. fingerprint, retina
  - does (dynamic biometrics) - e.g. voice, sign
- can use alone or combined
- all can provide user authentication
- all have issues

Password Authentication

- widely used user authentication method
  - user provides name/login and password
  - system compares password with that saved for specified login
- authenticates ID of user logging and
  - that the user is authorized to access system
  - determines the user’s privileges
  - is used in discretionary access control

Password Vulnerabilities

- offline dictionary attack
  - **Attack:** the attacker has the hash of the target password and he tries to break it
    - common passwords
    - Info related to the target
  - **Countermeasure**:

Password Vulnerabilities

- offline dictionary attack
  - **Attack:** the attacker has the hash of the target password and he tries to break it
    - common passwords
    - Info related to the target
  - **Countermeasure**:
    - protect these informations

Password Vulnerabilities

- Specific account attack
  - **Attack:** the attacker target a specific account and tries to guess the correct password
    - Common passwords
    - Info related to the target
  - **Countermeasure**:

Password Vulnerabilities

- Specific account attack
  - **Attack:** the attacker target a specific account and tries to guess the correct password
    - Common passwords
    - Info related to the target
  - **Countermeasure**:
    - account lockout mechanisms (i.e., allow only few authentication attempts)

Password Vulnerabilities

- Popular Password Guessing
  - **Attack**: the attacker tries popular password against a wide range of accounts
    - Users tend to choose simple passwords
    - Likely to detect some passwords
  - **Countermeasure**:

Password Vulnerabilities

- Popular Password Guessing
  - **Attack**: the attacker tries popular password against a wide range of accounts
    - Users tend to choose simple passwords
    - Likely to detect some passwords
  - **Countermeasure**:
    - Policies that do not allow the use of simple and common passwords

Password Vulnerabilities

- Workstation hijacking
  - **Attack**: The attacker waits until a logged-in workstation is unattended
  - **Countermeasure**:

Password Vulnerabilities

- Workstation hijacking
  - **Attack**: The attacker waits until a logged-in workstation is unattended
  - **Countermeasure**:
    - Automatically logging-out mechanisms
    - Anomaly behaviour detection

Password Vulnerabilities

- Exploiting user mistakes
  - **Attack**: Users tend to write down passwords
    - E.g., post-it near the protected device
    - Devices with pre-configured passwords
  - **Countermeasure**:

Password Vulnerabilities

- Exploiting user mistakes
  - **Attack**: Users tend to write down passwords
    - E.g., post-it near the protected device
    - Devices with pre-configured passwords
  - **Countermeasure**:
    - User training
    - Combined authentication mechanism
      - Password + token

Password Vulnerabilities

- Exploiting multiple password uses
  - **Attack**: users tend to use same (or similar) passwords in different systems
    - If an attacker correctly guess a password, he can extend the damage in multiple systems
  - **Countermeasure**:

Password Vulnerabilities

- Exploiting multiple password uses
  - **Attack**: users tend to use same (or similar) passwords in different systems
    - If an attacker correctly guess a password, he can extend the damage in multiple systems
  - **Countermeasure**:
    - User training
    - Forbid the password-reuse in multiple systems
      - Feasible only on a specific network that we can control

Password Vulnerabilities

- Electronic monitoring
  - **Attack**: if the password is communicated through a network, an attacker can sniff these packets and steal the password
  - **Countermeasure**:

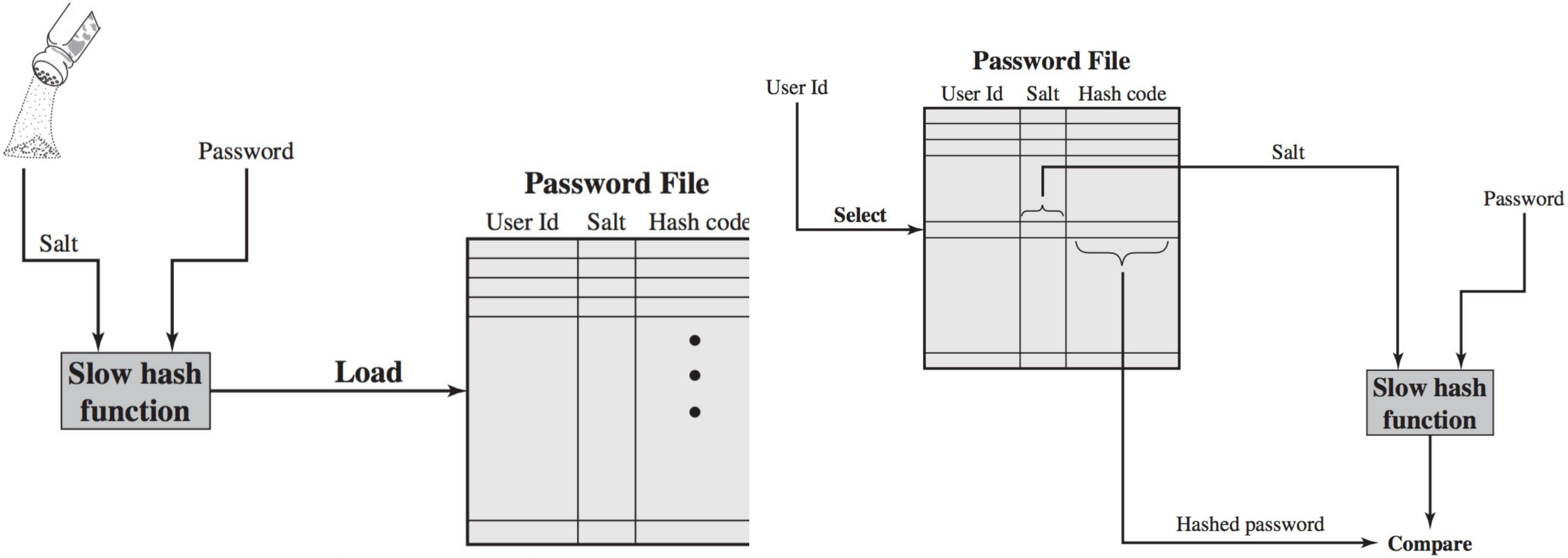
Password Vulnerabilities

- Electronic monitoring
  - **Attack**: if the password is communicated through a network, an attacker can sniff these packets and steal the password
  - **Countermeasure**:
    - Secure communication links

Hashed Passwords

- Widely used security mechanism
- Steps:
  - The user create a new **password**
  - The password is combined with a fixed length **salt**
    - The salt usually is pseudo-randomly generated
  - Hashed Password = Hash(password, salt)
  - ID, Hashed password and Salt are saved in a file
    - Password file
- These hashed functions are designed to be **slow**

Hashed Passwords



(a) Loading a new password

(b) Verifying a password

Hashed Passwords

- Why do we use salt?
  - This is not a chicken
- We can identify three main reasons:

Hashed Passwords

- Why do we use salt?
  - This is not a chicken
- We can identify three main reasons:
  - Password duplication prevention
    - If two users share the same password, the use of different salt produce different hashed passwords
  - Increase the difficultness of dictionary attacks
    - If the salt has *b* bits, the factor will be *2^b*
  - Impossible to find out if a person uses the same password in different systems

Password Cracking

- dictionary attacks
  - try each word then obvious variants in large dictionary against hash in password file
    - First try all common password
    - If there is no-matches, we try possible modifications (numbers, punctuation)
    - Computationally expensive
- rainbow table attacks
  - precompute tables of hash values for all salts
  - a mammoth table of hash values
  - e.g. 1.4GB table cracks 99.9% of alphanumeric Windows passwords in 13.8 secs
  - not feasible if larger salt values used

Password Choices

- ➢ users may pick short passwords
  - e.g. 3% were 3 chars or less, easily guessed
  - system can reject choices that are too short
- ➢ users may pick guessable passwords
  - so crackers use lists of likely passwords
  - e.g. one study of 14000 encrypted passwords guessed nearly 1/4 of them
  - would take about 1 hour on fastest systems to compute all variants, and only need 1 break!

Password File Access Control

- can block offline guessing attacks by denying access to encrypted passwords
  - make available only to privileged users
  - often using a separate shadow password file
- still have vulnerabilities
  - exploit O/S bug
  - accident with permissions making it readable
  - users with same password on other systems
  - access from unprotected backup media
  - sniff passwords in unprotected network traffic

Password Selection Strategies

- clearly have problems with passwords
- goal to eliminate guessable passwords
- If we cannot trust users … then we can guide them
- techniques:
  - user education -> they can ignore it
  - computer-generated passwords -> difficult to remember
  - reactive password checking -> require resources
  - proactive password checking -> likely the best solution

Proactive Password Checking

- rule enforcement plus user advice, e.g.
  - 8+ chars, upper/lower/numeric/punctuation
  - may not suffice
- password cracker
  - time and space issues
- Markov Model
  - generates guessable passwords
  - hence reject any password it might generate
- Bloom Filter
  - use to build table based on dictionary using hashes
  - check desired password against this table

Proactive Password Checking

- Bloom Filter
  - use to build table based on dictionary using hashes
  - check desired password against this table
- Bloom Filter mechanism
  - *K* order: *K* hash functions defined in a range [0, …, N-1]
  - Given a password, it calculates *k* hashed passwords
  - Maintain a Table of Size N
    - Table[hash] = 1 over the words of the common words dictionary
  - Given a new password
    - It calculates the k-hashes
    - The password is rejected if all the corresponding bits of the hash table are equal to 1

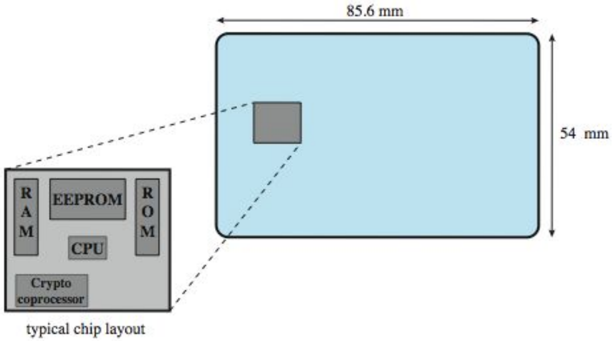
Token Authentication

➢ object user possesses to authenticate, e.g.
  - embossed card (e.g., old credit cards)
  - magnetic stripe card (e.g., hotel keys)
  - memory card (e.g., SIM)
  - Smartcard (e.g., Biometric ID card)

Memory Card

- store but do not process data
- magnetic stripe card, e.g. bank card
- electronic memory card
- used alone for physical access
- with password/PIN for computer use
- drawbacks of memory cards include:
  - need special reader (increase the cost of the security solution)
  - loss of token issues (we cannot trust users)
  - user dissatisfaction (not totally approved by users)

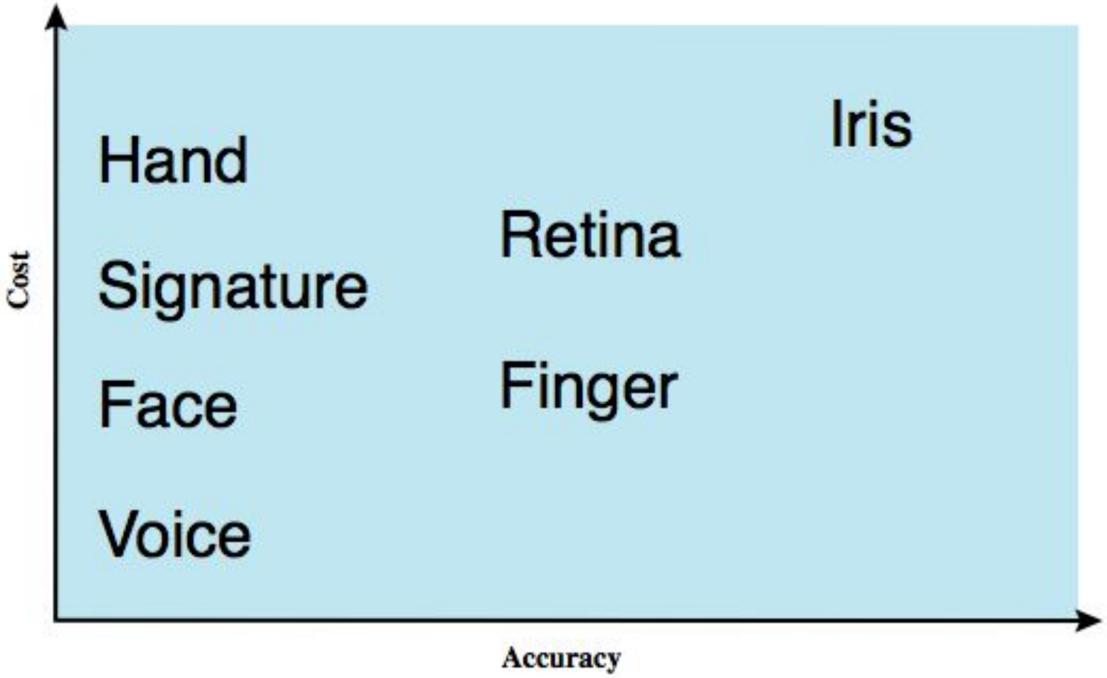
Smartcard



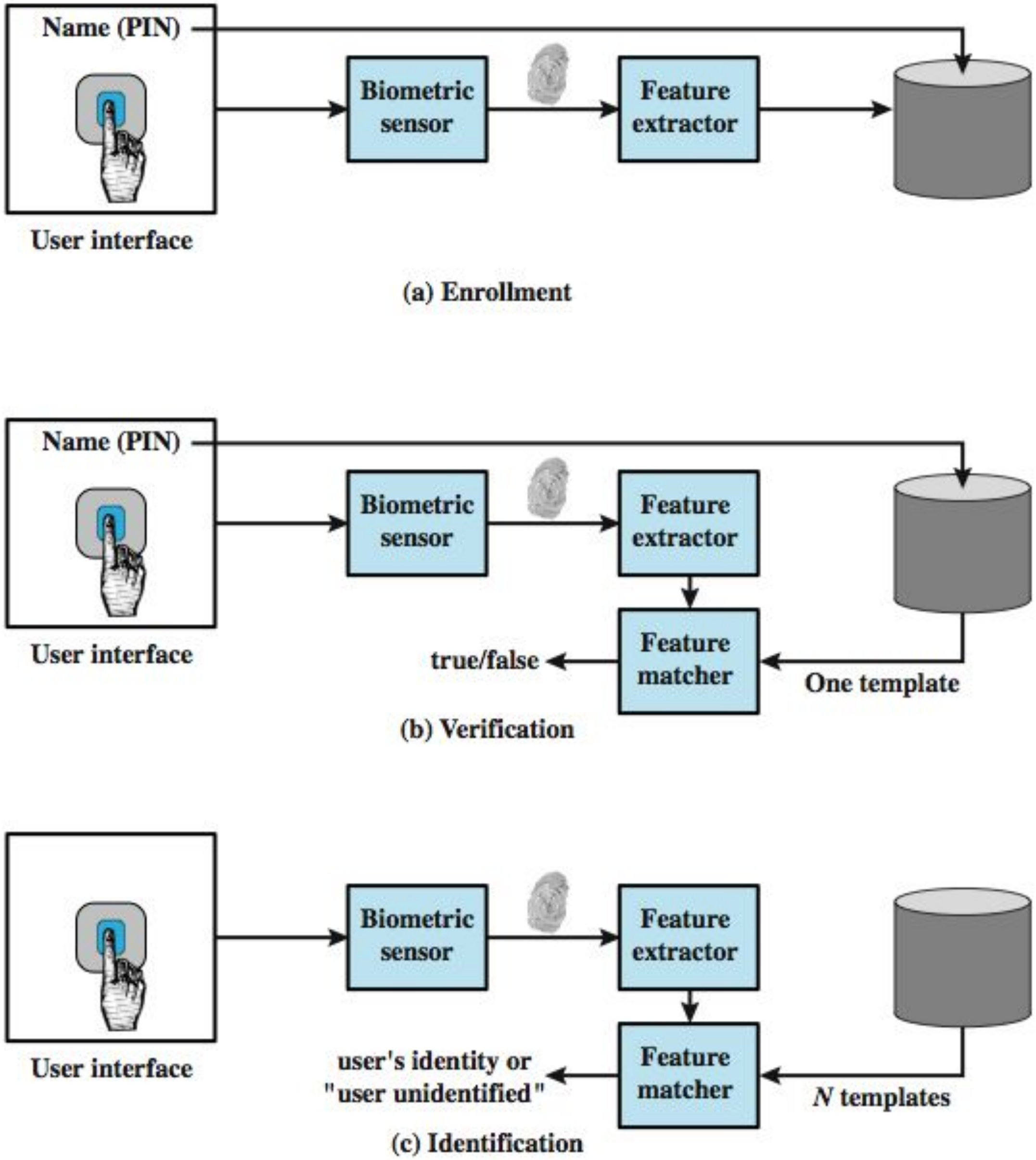
- credit-card like
- has own processor, memory, I/O ports
  - wired or wireless access by reader
  - may have crypto co-processor
  - ROM, EEPROM, RAM memory
- executes protocol to authenticate with reader/computer
- Another example is the USB dongle

Biometric Authentication

- authenticate user based on one of their physical characteristics



Operation of a Biometric System



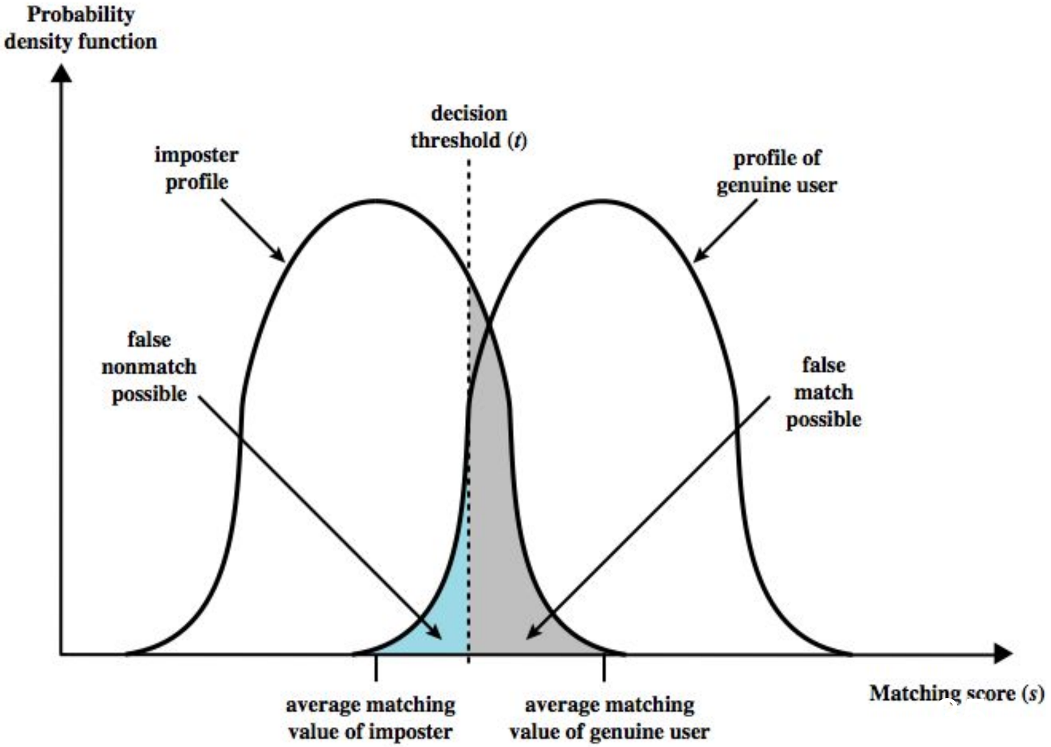
(a) Enrollment

(b) Verification

(c) Identification

Biometric Accuracy

- ➢never get identical templates
- ➢problems of false match / false non-match



Remote User Authentication

- Two type of authentications:
  - Local and from remote
- authentication over network more complex
  - problems of eavesdropping, replay
- generally use challenge-response
  - user sends identity
  - host responds with:
    - random number *r* (a.k.a. nonce)
    - An hash function *h*
    - A function *f*
  - user computes f(r,h(P)) and sends back
  - host compares value from user with own computed value, if match user authenticated
- protects against a number of attacks

Remote User Authentication

- Two type of authentications:
  - Local and from remote
- authentication over network more complex
  - problems of eavesdropping, replay
- generally use challenge-response
  - user sends identity
  - host responds with:
    - random number *r* (a.k.a. nonce)
    - An hash function *h*
    - A function *f*
  - user computes f(r,h(P)) and sends back
  - host compares value from user with own computed value, if match user authenticated
- protects against a number of attacks

Authentication Security Issues

- client attacks
  - adversary attempts to masquerade as a legitimate user (e.g., exhaustive search)
- host attacks
  - Attack to the host which contains hashed passwords
- Eavesdropping
  - Learn the password (e.g., by observing the user)
- Replay
  - Repetition of previously captured user response
- trojan horse
  - Malicious application that register the password
- Denial-of-service
  - Disable the user authentication by flooding the service

Summary

- introduced user authentication
  - using passwords
  - using tokens
  - using biometrics
- remote user authentication issues