Authentication

Week 5, Thursday

Authentication Basics

- Authentication is the binding of an identity to a subject, which is acting on behalf of an entity
  - Or, the binding of an identity to an entity

- How?
  - What the entity knows (e.g. passwords)
  - What the entity has (e.g. a badge)
  - What the entity is (e.g. fingerprints)
  - Where the entity is (e.g. in front of a particular terminal)

Authentication Process

- Obtain authentication info from an entity
- Analyze the info
- Determine whether or not the info is associated with the entity
- For the purpose of analysis, the entity’s info must be stored and managed
  - An authentication system

Authentication System

- \( A \): the set of authentication info with which entities prove their identities
- \( C \): the set of complementary info that the system stores and uses to validate the authentication info
- \( F \): the set of complementation functions that for \( f \in F, f: A \rightarrow C \)
- \( L \): the set of authentication functions that for \( l \in L, l: A \rightarrow \{ \text{true}, \text{false} \} \)
- \( S \): the set of selection functions that enable an entity to create/alter authentication and complementary info

Example

- A user authenticates himself using a password
- The system compares with the cleartext password stored online
- \( A = \{ s \mid s \text{ is an eligible password string} \} \)
- \( C = A \)
- \( F = \{ f \}, f \text{ is the identity function} \)
- \( L = \{ \text{eq} \}, \text{eq}(x, y) = \text{true}, \text{if } x = y; \text{false}, \text{otherwise} \)
- \( S = \{ u \mid u \text{ is a function to change password} \} \)

Passwords

- A password is information associated with an entity that confirms the entity’s identity
  - Simplest example: some sequence of characters
    - e.g., login, su, etc. in Unix
  - \( C \) may not be the same as \( A \)
    - Mostly because \( C \) must be protected
      - e.g., /etc/passwd (or shadow password files) in Unix
  - \( F \)
    - \( f \in F \) is based upon DES in Unix
  - \( S \)
    - e.g., passwd command in Unix
Attacking a Password System

- **Dictionary attack**: password guessing through repeated trial and error
  - Dictionary: a list of words for guesses
- **Type 1**:
  - For each guess $g$, compute $f(g)$ for each $f \in F$
  - If $f(g)$ is entity $E$'s complementary info, then $g$ authenticates $E$ under $f$
  - e.g., grab /etc/passwd file and then guess
- **Type 2**:
  - For guess $g$, if $l$ returns true, $g$ is correct password
  - e.g., try to log in as root in Unix

Countering Password Guessing

- **Anderson’s Formula**
- **Random selection of passwords**
- **Pronounceable passwords**
- **User selection of passwords**
- **Guessing through authentication functions**

**Anderson’s Formula**

- $P$: the probability that an attacker can successfully guess a password in a given period $T$
- $G$: # of guesses per time unit
- $N$: total # of possible passwords
- $P \geq TG/N$ ("=" when purely random guess)

**Random Selection of Passwords**

- Let the expected time required to guess a password be $T$. Then $T$ is a maximum when the selection of any of a set of possible passwords is equiprobable
- Sometimes randomness is not achieved
  - Bad random number generator used for password creation
  - Human psychological factors

**Pronounceable Passwords**

- Pronounceable: based on phoneme
  - “belgoret” “jutelone”
- But making type 1 dictionary attack easier
  - Smaller set of possible passwords for same length
  - Unless you use longer password
- **Key crunching**: hash a string of length $n$ or less to another string of length $k$ or less, where $n \geq k$.
  - MD5 or SHA-1 can be used for key crunching
User Selection of Passwords

- Proactive password selection/checker
  - Enable users to select
  - But enforces specific restrictions on password selection
- Klein’s study: what are bad passwords?
  - dictionary words, common names, user account names, phrases, make names, female names, uncommon names, machine names, place names, etc.
- Recommendation: \( \geq 1 \) digit, \( \geq 1 \) letter, \( \geq 1 \) punctuation symbol, \( \geq 1 \) control letter
  - “g0^A@4!”

Guessing Through Authentication Functions

- Recall that this is type 2 dictionary attack
- Cannot be prevented since the authentication must be available
  - Legitimate users need it
- But can try best to defend
  - Backoff (exponentially)
  - Disconnection
  - Disabling (if repeated failures are detected)
  - Jailing

Password Aging

- If C, F, L have not changed by the time the passwords is guessed, the attacker succeeds.
- **Password aging** requires that a password be changed after some period of time or after some event

Challenge-Response

- Fundamental flaw of passwords: reusability
  - Can be replayed if known before
  - What if every time one uses different authentication information
- In a challenge-response authentication system
  - User \( U \) and System \( S \) share a secret function \( f \)
  - \( S \) sends a random message \( m \) (challenge)
  - \( U \) replies with \( r = f(m) \) (response)
  - \( S \) validates \( r \) by computing it separately

Pass Algorithms

- The \( f \) in the previous slide is called **pass algorithm**
  - No keys or other secret info may be input to \( f \)
  - \( f \) itself is secret

One-Time Password

- **One-time password**: a password that is invalidated as soon as it is used
- Also a challenge-response mechanism
  - Challenge: the number of authentication attempt
  - Response: the one-time password
S/Key

- \( h \): a one-way hash function
- \( k \): an initial seed chosen by the user

Keys:
\[
h(k_0) = k_1, \ h(k_1) = k_2, \ldots, \ h(k_{n-1}) = k_n
\]
Passwds:
\[
p_i = k_0, \ p_2 = k_{n-1},\ldots, \ p_{n-1} = k_2, \ p_n = k_1
\]

If Eve intercepts \( p_i \), we know \( p_i = h(p_{i+1}) \), and \( h \) is a one-way hash function, so \( p_{i+1} \) cannot be derived from \( p_i \).

S/Key Authentication Protocol

- User Matt supplies his name to the server
- The server replies with the number \( i \) stored in the skeykeys file
- Matt supplies password \( p_i \)
- Server computes \( h(p_i) \) and compares it with the stored password \( p_{i-1} \). If match,
  - Authentication succeeds
  - \( i \leftarrow i+1, p_{i-1} \rightarrow p_i \)

Note: errors on page 326

Hardware-Supported Challenge-Response Procedures

- **Token device**
  - System sends a challenge
  - User enters it into the device (PIN maybe needed)
  - The device returns a response, by hashing (or enciphering) the challenge
  - The user sends the response over

(cont’d)

- **Temporally based device**
  - Every 60 seconds, a different number displayed
  - The system knows what number to be displayed for a user
  - When the user logs in, he enters the number currently shown
    - Followed by a fixed password
    - e.g., RSA SecureID card

Biometrics

- As old as humanity
- Fingerprints
- Voices
- Eyes
- Faces
- Keystrokes
- Combinations

Location

- Anna is logging from Russia
  - But we know she is now working at California
- Dennis and MacDoran’s scheme: use Global Positioning System (GPS)
  - An entity obtains a **location signature** using GPS
  - Transmits it
  - The System uses a **location signature sensor** (LSS) to obtain a similar location signature
  - Compare the two signatures to authenticate
Homework 4
(Due 5/6)

• If Alice sends Bob a message $m$ with the digital signature $(h(m))_d_{Alice}$, how would Bob verify the signature step by step?
• Case study 2 in digital signature. For a given message $m$ from Alice, Eve won’t be able to forge Alice’s signature $(m)_d_{Alice}$. However, Eve can forge a junk message and have it as if signed by Alice. Explain why.
• Chapter 11
  [page 307] 11.8 : 1, 2, 4
• Chapter 12
  [page 335] 12.10 : 2.a, 11, 14