Authentication

Murat Kantarcioglu

Authentication Overview

- Basics
- Passwords
- Challenge-Response
- Biometrics
- Location
- Multiple Methods

Basics

- There exists two reasons for authenticating users:
  - The user identity is a parameter in access control decisions
  - The user identity is recorded when logging security-relevant events in the audit trail
- It is not always necessary or desirable to base access control on user identities, while there is a much stronger case for using identities in the audit logs

Basics

- When a user connects to a computer system is has to enter
  - *User name* – this step is called *identification*
  - *Password* – this step is called *authentication*
- <u>Authentication</u>: the process of verifying a claimed identity

Verifying Identity

- One or more of the following
  - What entity knows (*eg.* password)
  - What entity has (*eg.* badge, smart card)
  - What entity is (*eg.* fingerprints, retinal characteristics)
  - Where entity is (*eg*. In front of a particular terminal)
  - Recent one
    - Who the entity knows? (e.g., references.)

Authentication Process

- It consists of several steps:
  - Obtaining the authentication information from an entity
  - Analyzing the data
  - Determining if the authentication information is associated with that entity

Authentication System

- (A, C, F, L, S)
  – A : information that proves identity
  – C : information stored on computer and used to validate authentication information
  – F : complementation function $f : A \rightarrow C$
  – L : functions that prove identity
  – S : functions enabling entity to create, alter information in A or C

Example

- Password system, with passwords stored on line in clear text
  - *A:* set of strings making up passwords
  - *C* = *A*
  - *F :* singleton set of identity function { *I* }
  - *L* : single equality test function { *eq* }
  - *S* : function to set/change password

Passwords

- Sequence of characters
  - Examples: 10 digits, a string of letters, *etc*.
  - Generated randomly, by user, by computer with user input
- Sequence of words
  - Examples: pass-phrases

  Note: A *pass-phrase* is a sequence of characters that it is too long to be a password and it is thus turned into a shorter virtual password by the password system
- Algorithms
  - Examples: challenge-response, one-time passwords

Storage

- Store as cleartext
  - If password file compromised, *all* passwords are revealed
- Encipher file
  - Need to have encryption, decryption keys in memory
  - Reduces to previous problem
- Store one-way hash of password
  - If file read, attacker must still guess passwords or invert the hash

Example

- UNIX system standard hash function
  - Hashes password into 11 char string using one of 4096 hash functions
- As authentication system:
  - *A* = { strings of 8 chars or less }
  - *C* = { 2 char hash id || 11 char hash }
    - The 2 char identify the hash function used
  - *F* = { 4096 versions of modified DES }
  - *L* = { *login*, *su*, … }
  - *S* = { *passwd*, *nispasswd*, *passwd+*, … }

Passwords-based Authentication

- A *password* is information associated with an entity that confirms its identity.
- How can passwords be protected?
- A solution: *one-way hashing*
  - A user's password is hashed and then stored. The stored password is never decrypted.
  - It should be difficult for an attacker to revert the stored password to the plaintext password.
  - A user A may try to guess the password of another user, B, and thus *impersonate* B. (next slide)

Analysis of an Impersonation Attack

- Goal: find $a \in A$ such that:
  - For some $f \in F$, $f(a) = c \in C$
  - c is associated with the given entity
- Two ways to determine whether a meets these requirements:
  - Direct approach: as above – it is possible if C is known to the attacker
  - Indirect approach: as $l(a)$ succeeds iff $f(a) = c \in C$ for some c associated with an entity, compute $l(a)$

Preventing Attacks

- Hide one of *a*, *f*, or *c*
  - Prevents obvious attack from above
  - Example: UNIX/Linux shadow password files
    - Hides *c*'s
    - Unix shadow password files can only be accessed by the super-user (access control is thus used)
- Block access to all $l \in L$ or result of $l(a)$
  - Prevents attacker from knowing if guess succeeded
  - Example: preventing *any* logins to an account from a network
    - Prevents knowing results of *l* (or accessing *l*)

Dictionary Attacks

- Trial-and-error from a list of potential passwords
  - Type 1: attacker knows *A*, *f*, *c*
    - Also referred to as *Off-line*: the attacker knows *f* and *c*'s, and repeatedly tries different guesses $g \in A$ until the list is done or passwords guessed
  - Type 2: attacker knows *A*, *l*
    - Also referred to as *On-line*: the attacker has access to functions in *L* and tries guesses *g* until some $l(g)$ succeeds
    - Examples: trying to log in by guessing a password

Approaches: Password Selection

- Random selection
  - Any password from *A* equally likely to be selected
  - Such passwords are difficult to remember for users, especially when they have multiple randomly-selected passwords
- Pronounceable passwords
- User selection of passwords

Pronounceable Passwords

- Generate phonemes randomly
  - Phoneme is unit of sound, eg. *cv*, *vc*, *cvc*, *vcv* where
    - *c* is a consonant
    - *v* is a vowel
  - Examples: helgoret, juttelon are pronounceable; przbqxdfl, zxrptglfn are not pronounceable
- Problem: the number of pronounceable passwords of length *n* is considerably lower than the number of random passwords of length *n*

User Selection

- Problem: people pick easy to guess passwords
  - Based on account names, user names, computer names, place names
  - Dictionary words (also reversed, odd capitalizations, control characters, “elite-speak”, conjugations or declensions, swear words, Torah/Bible/Koran/… words)
  - Too short, digits only, letters only
  - License plates, acronyms, social security numbers
  - Personal characteristics or foibles (pet names, nicknames, job characteristics, *etc*.)

Selecting Good Passwords

- Good passwords can be constructed in several ways
  - A password containing at least one digit, one letter, one punctuation symbol, and one control character is usually a strong password
- “LlMm*2^Ap”
  - Letters chosen from the names of members of 2 families
- “OoHeO/FSK”
  - Second letter of each word of length 4 or more in third line of third verse of Star-Spangled Banner, followed by “/”, followed by author’s initials

Proactive Password Checking

- Analyze proposed password for “goodness”
  - Always invoked
  - Can detect, reject bad passwords for an appropriate definition of “bad”
  - Discriminate on per-user, per-site basis
    - For example a password UTD$MK3 is not good at UTD.
    - Spell checker, for example
  - Easy to set up and integrate into password selection system

Example: OPUS System *

- Goal: check passwords against large dictionaries quickly
  - Run each word of dictionary through k different hash functions h_1, …, h_k producing values less than n
    - This is called Bloom filter.
  - Set bits h_1, …, h_k in OPUS dictionary
  - To check new proposed word, generate bit vector and see if *all* corresponding bits set
    - If so, word is in one of the dictionaries to some degree of probability
    - If not, it is not in the dictionaries

*: **OPUS: Preventing Weak Password Choices**
E. Spafford
http://www.cerias.purdue.edu/homes/spaf/tech-reps/9128.ps

Salting

- Goal: slow dictionary attacks aimed at finding *any* user’s password (as opposed to a *particular* user’s password)
- Method: perturb hash function so that:
  - Parameter controls *which* hash function is used
  - Parameter differs for each password
  - To determine if the string *s* is the password for any of a set of *n* users, the attacker has to perform *n* complementations, each of which generates a different complement

Guessing Passwords Through *L*

- If the actual complements, or the complementation functions, are not publicly available, the only way to try to guess a password is the use of the authentication function
- This attack cannot be prevented, otherwise, legitimate users cannot log in
- A solution is to make them slow
  - Backoff – the most common form is the exponential backoff
    - Let x be a parameter selected by the administrator; the system waits $x^0 = 1$ second before re-prompting the user; after n failures the system waits x^{n-1} seconds
  - Disconnection – it is effective when establishing connections is time-consuming (e.g. dialing a phone number)
  - Disabling
    - Be very careful with administrative accounts!
  - Jailing - Allow in, but restrict activities. It has interesting connections with access control

Password Aging

- Force users to change passwords after some time has expired
  - How do you force users not to re-use passwords?
    - Record previous passwords
    - Block changes for a period of time
  - Give users time to think of good passwords
    - Don’t force them to change before they can log in
    - Warn them of expiration days in advance

Challenge-Response

- Passwords have the fundamental problems that they are *reusable*
- If an attacker sees a password, she can later *replay* the password
- An alternative is to authenticate in such a way that the transmitted password changes each time
- Let a user u wishing to authenticate himself to a system S. Let u and S have an agreed-on secret function f. A *challenge-response* authentication system is one in which S sends a random message m (the challenge) to u, and u replies with the transformation $r = f(m)$ (the response). S then validates r by computing it separately.

Challenge-Response

- The user and system share a secret function *f* (in practice, *f* can be a known function with unknown parameters, such as a cryptographic key)

$$\textit{user} \xrightarrow{\textit{request to authenticate}} \textit{system}$$

$$\textit{user} \xleftarrow[\textit{(the challenge)}]{\textit{random message } r} \textit{system}$$

$$\textit{user} \xrightarrow[\textit{(the response)}]{f(r)} \textit{system}$$

Challenge-Response

Pass Algorithms

- Challenge-response with the function *f* itself a secret
  - Example:
    - Challenge is a random string of characters such as “abcdefg”, “ageksido”
    - Response is some function of that string such as “bdf”, “gkio”
    - The algorithm is every other letter beginning with the second
  - Can alter algorithm based on ancillary information
    - Network connection is as above, dial-up might require “aceg”, “aesd”
  - Usually used in conjunction with fixed, reusable password

Challenge-Response

Approaches based on cryptographic public keys

- Use of shared key could be problematic. Instead, PK could be used.
- Goal: *A* identifies *B* by checking whether *B* holds the secret key k_B that matches the public key K_B
- Assumptions: *A* chooses a random challenge (nonce) r_A. *B* uses its random nonce r_B. *B* applies its public-key system for generating a signature.
- Message sequence:

1. $A \rightarrow B$: r_A.
2. $B \rightarrow A$: r_B, $\mathrm{Sign}_{k_b}(r_a, r_b)$

One-Time Passwords

- Password that can be used exactly *once*
  - After use, it is immediately invalidated
- Problems
  - Synchronization of user and system
  - Generation of good random passwords
  - Password distribution problem

S/Key

- One-time password scheme based on idea of Lamport
- *h* one-way hash function (MD5 or SHA-1, for example)
- User chooses initial seed *k*
- The key generator calculates:

$$h(k) = k_1,\ h(k_1) = k_2,\ \ldots,\ h(k_{n-1}) = k_n$$

- Passwords are in reverse order:

$$p_1 = k_n,\ p_2 = k_{n-1},\ \ldots,\ p_{n-1} = k_2,\ p_n = k_1$$

S/Key

- Suppose an attacker intercepts p_i.
- Because $p_i = k_{n-i+1}$, $p_{i+1} = k_{n-i}$, and $h(k_{n-i}) = k_{n-i+1}$, we have that $h(p_{i+1}) = p_i$
- Thus, the attacker in order to guess p_{i+1} from p_i would have to invert h; because h is a one-way function, it will be hard to invert

S/Key Protocol

System stores maximum number of authentications n, number of next authentication i, last correctly supplied password p_{i-1}.

$$\textit{user} \xrightarrow{\{\ \textit{name}\ \}} \textit{system}$$

$$\textit{user} \xleftarrow{\{\ i\ \}} \textit{system}$$

$$\textit{user} \xrightarrow{\{\ p_i\ \}} \textit{system}$$

System computes $h(p_i) = h(k_{n-i+1}) = k_{n-i} = p_{i-1}$. If match with what is stored, system replaces p_{i-1} with p_i and increments i.

Biometrics

- Automated measurement of biological, behavioral features that identify a person
  - Fingerprints: optical or electrical techniques
    - Maps fingerprint into a graph, then compares with database
    - Measurements imprecise, so approximate matching algorithms used
  - Voices: speaker verification or recognition
    - Verification: uses statistical techniques to test hypothesis that speaker is who is claimed (speaker dependent)
    - Recognition: checks content of answers (speaker independent)

Other Characteristics

- Can use several other characteristics
  - Eyes: patterns in irises unique
    - Measure patterns, determine if differences are random; or correlate images using statistical tests
  - Faces: image, or specific characteristics like distance from nose to chin
    - Lighting, view of face, other noise can hinder this
  - Keystroke dynamics: believed to be unique
    - Keystroke intervals, pressure, duration of stroke, where key is struck
    - Statistical tests used

Location

- If you know where user is, validate identity by seeing if person is where the user is
  - Requires special-purpose hardware to locate user
    - GPS (global positioning system) device gives location signature of entity
    - Host uses LSS (location signature sensor) to get signature for entity

Multiple Methods

- Example: “where you are” also requires entity to have LSS and GPS, so also “what you have”
- Can assign different methods to different tasks
  - As users perform more and more sensitive tasks, must authenticate in more and more ways (presumably, more stringently) File describes authentication required
    - Also includes controls on access (time of day, *etc*.), resources, and requests to change passwords
  - Pluggable Authentication Modules

Key Points

- Authentication is not cryptography
  - You have to consider system components
- Passwords are here to stay
  - They provide a basis for most forms of authentication
- Protocols are important
  - They can make masquerading harder
- Authentication methods can be combined