On Authentication

“I was so frightened, I couldn't remember my own name!”
White Queen in Through the Looking Glass, by Lewis Carroll

Password authentication

• User authenticates herself by entering a password -- who is checked against the server's database
  – Pros:
    • Supported by almost every system
    • Users familiarized with the process
  – Cons:
    • Good password management requires more of the user than other methods
Unix Password System

- User-entered passwords are converted into bit strings.
- A salt (set by the system during user account creation and stored in the account database) is also used as input.
- A hash of the password and salt are computed, the value encoded in printable ASCII characters, and stored in the account database.
- The hash algorithm varies. In early systems, the bit-string for the password is treated as a key for (a variant of) DES. The salt is used to indicate which DES variant to use (a salt all of its bits are 0 result in DES being selected).
- In modern systems, the hash is either based on the hash function MD5, or on the Blowfish block cipher.

Features of Unix passwords

- The user must know the password to authenticate him/herself
- That creates problems for remote authentication---the password must be sent over the network connection. This requires extraneous methods to protect the connection.
- If the hashed password and salt are obtained (say, by compromising the user database), it is possible to perform dictionary attacks to recover the password.
Dictionary attacks

- A word list is compiled
- Dictionary entries (often in several languages), as well as common proper names, are included in such lists.
- Often the list is expanded to include some simple variations, such as “b1rd” or “hell0”.
- Online dictionary attack: try to login with each password in the dictionary.

Offline dictionary attacks

- Offline dictionary attack: All the entries of the word list are hashed, and the result is stored in a database.
  - Note that each word must be hashed with each possible salt value.
  - When the account database is compromised, the attacker can compare with the pre-computed data to break simple passwords in a very short time.
  - The use of salt can increase the pre-computation/storage requirements of attacks. But powerful time-memory trade-off algorithms are available to handle even large salt spaces, such as 64-bit salts.
- Brute-force attack: For relatively short passwords, all possible variations can be pre-computed and storage, using the same time-memory trade-off strategies.
Mitigation strategies

• Force users to choose good passwords
  – Specify need to include lower and upper-case alphabet characters, digits and non-alphanumeric keys.
  – Checking them against dictionary attacks
• Make users to change passwords often
• Use salted passwords (note that the salt does not solve the problem of poor password choices, only makes attacks against reasonably good passwords more costly).

Password-based Challenge/Response

• Some systems enable secure remote authentication via passwords by using challenge-response methods.
• The system generates a random challenge C
• The user returns a function \( F(C, H) \), where H is the hash of his/her password
  – The user enters the password in the remote system, which re-computes the hash H from it, and then to compute $F(C, H)$. 
Features of challenge-response authentication

• Compromise of the account database of hashed password enable remote authentication without having to guess the password
  – Since the server only knows the hash, not the password, the mechanism must use the hash directly

• Vulnerable to offline dictionary and brute-force attacks on the password by eavesdropping on the transmitted values F(C, H).
  – Note that C is sent in the clear, so the adversary does not even need to compromise the account database to capture values and mount an offline attack.

Strong Authentication

“How am I to get in?” asked Alice again, in a louder tone. “ARE you to get in at all?” said the Footman. “That's the first question, you know.”

From Alice in the Wonderland
First Steps

• How can we make this protocol secure?

• Note that Bob knows sensitive information about Alice, namely her password. What if we use a cryptographic key instead?

Challenge/Response Revisited

K = shared key between Alice and Bob
Distributing keys

• In a large network, it is infeasible to assume that each pair of nodes share a secret key.

• Idea: Use a central server to distribute keys, the **Key Distribution Center (KDC)**.
  – Analogy: In a LAN, often the password database is stored in a single server

Using a KDC

• Entity $\alpha$ contacts the KDC and uses $K_\alpha$ (shared key) to authenticate herself and request a key for use with entity $\beta$.
• KDC generates a session key $R_{\alpha\beta}$
• KDC encrypts the session key for $\alpha$ and for $\beta$
• KDC gives both copies to $\alpha$
• $\alpha$ decrypts the session key and contacts $\beta$, giving him the encryption of the session key computed by the KDC
A Global KDC infrastructure

- If each organization wishes to set up a KDC, it becomes infeasible for each to have a key shared with all the others
- KDCs need to authenticate other KDCs
  - Hierarchical infrastructure, with root KDC
  - Trust graphs, with varying trust levels
    - KDC1 does NOT trust KDC2
    - KDC1 trusts KDC2 to authenticate KDC2 users
    - KDC1 trusts KDC2 to authenticate any users
    - KDC1 trusts KDC2 to introduce trusted KDCs
Session Keys

• Session keys should be used for securing channels, with long-term key only for authentication
  – Reduces amount of information encrypted under the same key/ make cryptanalysis difficult
  – Prevents replay attacks
• Known-key attack:
  – Long-term keys derivable from session transcript + session key (BAD!)

More on Session Keys

• Leak of a session-key should NOT enable discovery of further session keys
  – Otherwise as bad as leaking the long-term key
• Forward secrecy:
  – Previous session keys undiscoverable after compromise of a session key
  – Previous session keys undiscoverable after compromise of a long-term key
  • This last level of security requires techniques from public key cryptography