User Authentication Basics

Authentication and Identity

• What is identity?
  – which characteristics uniquely identifies a person?
  – do we care if identity is unique?
• Authentication: verify a user’s identity
  – a supplicant wishes to authenticate
  – a verifier performs the authentication
• What’s relationship of identity to role, or job function?
User Authentication Can Be Based On…

1. What the user knows
   – passwords, personal information, a key, a credit card number, etc.
2. Where the user is or can be reached
   – email address, IP address, …
3. Physical characteristics of the user
   – fingerprints, voiceprint, signature dynamics, iris pattern, DNA, etc.
4. What the user has in their possession
   – smart card, (physical) key, USB token, …

Based on… (cont’d)

• Which of the above is best? best in what way?

Crypto-Based Authentication

• Basic idea: user performs a requested cryptographic operation on a value (a challenge) that the verifier supplies
• Usually based on knowledge of a key (secret key or private key)
• Examples: RSA, zero knowledge proofs, …
• we’ll look at such protocols in more detail next time
**Address-Based User Authentication**

- Associates identity with network address or email address
  - used by many web services
- Several early OS functions and tools worked this way
- Benefits? Problems?

**Password Authentication**

**Password-Based User Authentication**

- User demonstrates knowledge of a secret value to authenticate
  - most common method of user authentication
- Threats to password-based authentication?
Some Issues for Password Systems

• A password should be **easy** to remember but **hard** to guess
  – that’s difficult to achieve!
• Some questions
  – what makes a good password?
  – where is the password stored, and in what form?
  – how is knowledge of the password verified?

Password Storage

• Storing unencrypted passwords in a file is **high risk**
  – compromising the file system compromises all the stored passwords
• Better idea: use the password to compute a one-way function (e.g., a hash, an encryption), and store the *output of the one-way function*
• When user inputs the requested password…
  1. compute its one-way function
  2. compare with the stored value

Attacks on Passwords

• Suppose passwords could be up to 9 characters long
• This would produce $10^{18}$ possible passwords; **320,000 years** to try them all at 10 million a second!
• Unfortunately, not all passwords are equally likely to be used
Example of a Study

- In a sample of over 3000 passwords:
  - 500 were easily guessed versions of dictionary words or first name / last name
  - 86% of passwords were easily guessed

<table>
<thead>
<tr>
<th>Length in characters</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of passwords</td>
<td>15</td>
<td>72</td>
<td>464</td>
<td>477</td>
<td>706</td>
<td>605</td>
</tr>
</tbody>
</table>

(lower case only)

Common Password Choices

- Pet names
- Common names
- Common words
- Dates
- Variations of above (backwards, append a few digits, etc.)

Online Dictionary Attacks

- Can statically create a dictionary of all such choices
- Guess passwords using a dictionary
- Send guesses to verifier for confirmation or refutation

- Effective? Possible defensive techniques?
Offline Dictionary Attacks

- Obtain password file with hashed password values
- Guess passwords, compute hash, and compare with values stored in file
  - much faster than offline attacks!
- Effective? Possible defenses?

“Salting” the Password

- Many systems append to each password a random value (salt) before computing the one-way function
  - value is fixed and unique per user
  - stored with user’s hash value in the password file
- Increases the effort required for offline dictionary attacks
  - each bit of salt doubles the effort required
  - dictionaries with precomputed hashes of common passwords will not be useful
- Doesn’t increase resistance to online attacks?

Example: Unix Passwords

- Keyed password hashes are stored, with two-character (16 bit) salt prepended
  - password file is publicly readable
- Users with identical passwords but different salt values will have different hash values
Password Guidelines For Users

1. Initial passwords are system-generated, have to be changed by user on first login
2. User must change passwords periodically
3. Passwords vulnerable to a dictionary attack are rejected
4. User should not use same password on multiple sites
5. etc. etc.

Other Password Attacks

• Technical
  – eavesdropping on traffic that may contain unencrypted passwords (especially keystroke logging)
  – “trojan horse” password entry programs
  – man-in-the-middle network attack
• “Social”
  – careless password handling or sharing
  – phishing

The S/Key Protocol
Using “Disposable” Passwords

• Simple idea: generate a long list of passwords, use each only one time
  – attacker gains little/no advantage by eavesdropping on password protocol, or cracking one password

• Disadvantages
  – storage overhead
  – users would have to memorize lots of passwords!

• Alternative: the S/Key protocol
  – based on use of one-way (e.g. hash) function

S/Key Password Generation

1. Alice selects a password \( x \)
2. Alice specifies \( n \), the number of passwords to generate
3. Alice’s computer then generates a sequence of passwords
   \[ x_1 = H(x) \]
   \[ x_2 = H(x_1) \]
   \[ \ldots \]
   \[ x_n = H(x_{n-1}) \]

4. Alice communicates (securely) to a server the last value in the sequence: \( x_n \)
   • Key feature: no one knowing \( x_i \) can easily find an \( x_{i+1} \) such that \( H(x_{i+1}) = x_i \)
     – only Alice possesses that information
Authentication Using S/Key

• Assuming server is in possession of $x_i$ ...

Is dictionary attack still possible?

Limitations

• Value of $n$ limits number of passwords
  – need to periodically regenerate a new chain of passwords
• Does not authenticate server! Example attack:
  1. real server sends $i$ to fake server, which is masquerading as Alice
  2. fake server sends $i$ to Alice, who responds with $x_{i-1}$
  3. fake server then presents $x_{i-1}$ to real server

Biometrics
### Biometrics

<table>
<thead>
<tr>
<th>Desired qualities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. uniquely identifying</td>
</tr>
<tr>
<td>2. very difficult to forge / mimic</td>
</tr>
<tr>
<td>3. highly accurate, does not vary</td>
</tr>
<tr>
<td>4. easy to scan or collect</td>
</tr>
<tr>
<td>5. fast to measure / compare</td>
</tr>
<tr>
<td>6. inexpensive to implement</td>
</tr>
</tbody>
</table>

- Which of these are concerns for passwords?

### Assessment

<table>
<thead>
<tr>
<th>Convenient for users (e.g., you always have your fingerprints, never have to remember them), but…</th>
</tr>
</thead>
<tbody>
<tr>
<td>- potentially troubling sacrifice of private information</td>
</tr>
<tr>
<td>- no technique yet has all the desired properties</td>
</tr>
</tbody>
</table>

### Example Biometric Technologies

<table>
<thead>
<tr>
<th>Example Biometric Technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature / penmanship / typing style</td>
</tr>
<tr>
<td>Fingerprints</td>
</tr>
<tr>
<td>Palm geometry</td>
</tr>
<tr>
<td>Retina scan</td>
</tr>
<tr>
<td>Iris scan</td>
</tr>
<tr>
<td>Face recognition</td>
</tr>
<tr>
<td>Voice recognition</td>
</tr>
</tbody>
</table>
Multifactor Authentication

• If one characteristic is pretty good, two or more characteristics should be better?

• Suppose true positive rate was AND of the two, and false positive rate was OR of the two…
  – TP = TP1 * TP2
  – FP = 1 - (1-FP1)*(1-FP2)

• Alternative: combine a biometric technique with passwords

Authentication Hardware (Tokens)
Tokens

- A token is a physical device that can be interfaced to the computer, and carries identifying information
- Types
  - **passive** tokens just store information
  - **active** tokens have processors and can perform cryptographic operations
- Examples
  - cards with magnetic strips
  - smart cards
  - USB storage devices
  - RFID tags

Design Issues for Tokens

- Cost
- Size
- Capabilities
- Robustness
- Resistance to tampering
- Usefulness if stolen / lost

An Example: *Time Synchronized* Tokens

- The token contains:
  - *internal clock*
  - display
  - a secret key
- Token computes a one-way function of current time+key, and displays that
  - this value changes about once per minute
- User reads this value and types it in to authenticate to the server
  - requires that server and token time stays *synchronized*
Another Example: Alladin eToken

<table>
<thead>
<tr>
<th>API / standards</th>
<th>PKCS#11 v2.21, CAPI (Microsoft Crypto API), Siemens/Infineon APDU commands, PC/SC, X.509 v3 certificate storage, SSL v3, IPsec/IKE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Algorithms</td>
<td>RSA 1024-bit / 2048-bit*, DES, 3DES, SHA1</td>
</tr>
<tr>
<td>Power source</td>
<td>Battery, 5 year lifetime</td>
</tr>
<tr>
<td>LCD</td>
<td>6 characters</td>
</tr>
<tr>
<td>Data retention</td>
<td>10 years</td>
</tr>
</tbody>
</table>

Summary

1. Passwords are by far the most widely used form of authentication, despite numerous problems
2. Biometrics hold promise but are expensive, inconvenient, and compromise privacy
3. Two factor authentication is commonly used for higher security
4. One-time passwords (S/Key) are attractive, especially if combined with hardware