User Authentication Protocols

Week 7
Homework 1 is posted on the class webpage
Due in 2 weeks
10 points (out of 100) subtracted each late day
User Authentication

- The process of verifying an identity claimed by a system entity
- Fundamental system security building block
  - Basis of access control & user accountability
- Has two steps:
  - Identification – provide claimed identity
  - Authentication – verify validity of claim
- User authentication ≠ message authentication
User Authentication: How?

- Based on something the individual knows:
  - Knows - e.g. password, PIN
  - Possesses - e.g. key, token, smartcard
- Is (static biometrics): fingerprint, retina
- Does (dynamic biometrics): voice, handwriting
- Can use alone or combined
- All can provide user authentication
- All have issues
Authentication Protocols

- Convince parties of each others identity
  - Also exchange session keys
- May be one-way or two-way (mutual)

Key issues:

1. Confidentiality
   - Protect session keys
   - Prior keys or secrets need to exist

2. Timeliness
   - Prevent replay attacks
Replay Attacks

- Valid signed message is copied and later re-sent
- Simple replay
  - Copy message; replay later
- Repetition that can be logged
  - Replay timestamped message within validity interval
- Repetition that cannot be detected
  - Suppress original message
- Backward replay without modification
  - Send the replay message back to its sender
Replay Attacks: Countermeasures

- Sequence numbers
  - Attach sequence number \textit{seqno} to message
  - Accept message if \textit{seqno} follows previous value
  - Not always practical

- Timestamps
  - Message needs to contain \textit{timestamp}
  - Accept message if timestamp is within validity window
  - Need synchronized clocks
Countermeasures (cont’d)

- Challenge/response
  - Ensures message *freshness*
  - Challenger sends random nonce R
  - Responder’s message needs contain a function of R

![Diagram of Challenge/response protocol]

1. **Challenge**: $R$
2. **Response**: contains $F(R)$

*Alice* (Host)
Authentication

- One-way authentication
- Mutual: two-way authentication
  - Using symmetric key crypto
  - Using public-key crypto
One-Way Authentication

How can Trent T know it’s Alice and not Mallory impersonating Alice?
Authentication Approaches

- **Password**
  - Host stores Alice’s password
  - Alice sends password
  - Host verifies password

- **Problem:**
  - Trent stores all passwords in clear
  - Whoever breaks into Trent can steal passwords

- **Solutions**
  - One-Way Functions
  - Dictionary Attacks and Salts
Authentication Using Hashes

Roger Needham and Mike Guy
- T does not need to know password
- Only differentiate between valid and invalid ones

Alice
- Login A, pwd
- User ID
- Password file

Problem ?

Trent T (Host)
- Compare H(pwd) to H\text{A}

Password file
Password Vulnerabilities

- One-way hashes are vulnerable
  - Which password is better?
    - Barney
    - 9(hH/A.
  - Which one is easier to remember?
- Dictionary attack
  - Compile list of most probable passwords
  - Apply hash function to each
  - Compare against the password file
  - If match, password has been found!
## Defending with Salts!

### Salt: per user random value

![Alice](image1.png)

1. **Login A, pwd**

<table>
<thead>
<tr>
<th>User ID</th>
<th>salt</th>
<th>H(salt, pwd)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>s</td>
<td>$H_A$</td>
</tr>
</tbody>
</table>

**Password file**

![Trent T (Host)](image2.png)

$H(s, pwd) = H_A$
Example: Linux

- Passwords stored in /etc/shadow
  - Root readable only
- carbunar: $6!GHQQKZn$8.eJLvAaJiDTFAauGVbFlmnAcjIKyLtH6GiO0mVgra8weKJ1igU2BmgdDQAalynFQ0QuezQr7mDTWEPD7sDrW
- $6$: hash algorithm
  - $1$ = MD5 hashing algorithm.
  - $2$ = Blowfish Algorithm is in use.
  - $2a$ = eksblowfish Algorithm
  - $5$ = SHA-256 Algorithm
  - $6$ = SHA-512 Algorithm
Example: Linux

- Passwords stored in /etc/shadow
  - Root readable only
- `carbunar:$6$IGHQQKZn$8.eJLvAaJiDTFAauGVbFlmnAcjIKyLtH6GiO0mVgra8weKJ1igU2BmgdDQAalynFQ0QuezQr7mDTWEPD7sDrW`
  - salt
  - hash
The Goal of Salts

- Ensure that attacker cannot use the same dictionary to break all passwords

- Instead, attacker has to do a per-user dictionary + computation ...
Improved Dictionary Attack [D. Klein]

1. Copy the password file

2. For each user A with salt s and hash $H_A$
   1. Collect dictionary $D_A$ of tentative passwords
   2. Hash all items in $D_A$ using salt s
   3. Compare result against $H_A$

3. If match exists, found password

- 40% of passwords were guessed on average system!
Building the Dictionary

1. Name, initials, account name
   - Example: Daniel V. Klein, account – klone
   - klone0, klone1, ..., dvk, dklein, DKlein, dvklein, etc

2. Words from databases
   - Men and women names, nicknames (also famous)
   - Places
   - Variations of the above (capitalizations, plurals, etc)

3. Foreign language words

4. Word pairs
Conclusions

- Never use your personal information
- Do not use words (dictionary)
- Use combination of words and characters
- Do not use same passwords for all systems
- Change your password frequently
- Use passphrases
- Example:
  - ”My Password is not easy to crack”
  - mpine2C.
**Alternative: SKEY**

*Use hash-chains*

1. **Init, A, \(x_{100}\)**

2. **Login, A, \(x_{99}\)**

3. **Login, A, \(x_{98}\)**

---

**Alice**

Generate \(R\)

Compute

\[x_1 = H(R)\]

\[x_2 = H^2(R) = H(H(R))\]

\[x_3 = H^3(R) = H(H(H(R)))\]

...  

\[x_{100} = H^{100}(R)\]
Authentication

- One-way authentication
- Mutual: two-way authentication
  - Using symmetric key crypto
  - Using public-key crypto
What is Mutual Authentication?

1. Authenticate
1'. Exchange keys

Bob B

Alice

Mallory

Make sure they don’t talk to Mallory!
Authentication

- One-way authentication
- Mutual: two-way authentication
  - Using symmetric key crypto
  - Using public-key crypto
Using Symmetric Keys

1. Exchange keys
1'. Authenticate

Assume T shares a key with A ($K_A$) and B ($K_B$)

$E_A(M)$: encryption with key shared by A and T
Wide-Mouth Frog

Simplest Authentication/Key Exchange

1. Generate random $K$

2. $A, E_A(T_A, B, K)$

3. Decrypt message

4. $E_B(T_P, A, K)$

5. $E_K(M)$
Wide-Mouth Frog Observations

- Alice and Bob trust each other because of Trent
- **Timestamps prevent replay attacks (Why ?)**
- Trent is single point of failure/bottleneck
- **Assumption:**
  - Alice is able to generate good random numbers
Assume T shares a key with A ($K_A$) and B ($K_B$).

1. $A, R_A$
2. $B, E_B(A, R_A, R_B)$
3. Generate random $K$
4. $E_A(B, K, R_A, R_B)$
4'. $E_B(A, K)$
5. $E_B(A, K), E_K(R_B)$

Equal?
Yahalom Observations

- This time the protocol is initiated by B (not T)
- T chooses the key K to be shared by A and B
- A and B trust each other
  - Because of $R_A$ and $R_B$
  - Only T and B have access to $R_B$
- Problem in step 1 -- $R_A$ is sent in clear
  - Can Mallory impersonate B?
- No!
  - In step 4, T includes the identity of B - A will know who it is talking to
**Needham-Schroeder**

1. \(E_{A}(R_{A}, B, K, E_{B}(K,A))\)
2. Generate random \(K\)
3. \(E_{A}(R_{A}, B, K, E_{B}(K,A))\) \(\rightarrow\) \(E_{K}(R_{B}-1)\)
4. Extract key \(K\)
5. \(E_{B}(K,A)\)
6. Extract key \(K\)
7. Generate random \(R_{B}\)
8. \(E_{K}(R_{B})\)
9. \(E_{K}(R_{B}-1)\)

**Equal?**

**Trent T**

**Bob B**
Needham-Schroeder Observations

- **What is the purpose of $R_A$?**
  - For $A$ to prevent replay attacks
  - Ensure it is talking to $T$

- **What is the purpose of $R_B$?**
  - For $B$ to prevent replay attacks
  - And ensure that it is talking to $A$

- **Major weakness**
  - If Mallory gets hold of an old key $K$, it can impersonate $A$

- Solution: use timestamps
Otway-Rees

1. I, A, B, E\(_A(R_A, I, A, B)\)

2. I, E\(_A(R_A, K)\)

3. Generate random K

4. I, E\(_A(R_A, K)\), E\(_B(R_B, K)\)

5. Match?

I – index number

“I” needs to be the same across protocol!
Kerberos - Simplified

Kerberos 5: Variant of Needham-Schroeder

1. Generate timestamp t
2. Generate lifetime L
3. Generate random K
4. Trent T (Host)
5. $E_A(t, L, K, B), E_B(t, L, K, A)$
6. $E_K(A, t), E_B(t, L, K, A)$
7. $E_K(t+1)$
Kerberos Observations

- What is the goal of the timestamp and lifetime?
  - To prevent replay attacks
  - The messages are valid only in \([t, t+L]\)

- Major assumption:
  - The clocks are synchronized!
  - Not trivial (see Lamport’s clocks)

- In practice
  - Use time servers
  - Sync within a few minutes
Authentication

- One-way authentication
- Mutual: two-way authentication
  - Using symmetric key crypto
  - Using public-key crypto
Authentication with Public Keys

Assume T has a database of public keys for each participant.

- **pK_A**: A’s public key
- **E(pk_A, M)**: encryption with A’s public key
- **S_A(M)**: signature with A’s private key
Denning-Sacco

1. A, B

2. $S_T(B, pk_B), S_T(A, pk_A)$

3. Generate timestamp $T_A$

4. Generate random $K$

5. $E(pk_B, S_A(K, T_A)), S_T(B, pk_B), S_T(A, pk_A)$

6. Decrypt with its private key
   Verify A’s signature

7. Recover key $K$

Trent T
(Host)
Attacking Denning-Sacco!

1. $S_T(B, pk_B), S_T(C, pk_C)$
2. $E(pk_C, S_A(K, T_A)), S_T(C, pk_C), S_T(A, pk_A)$
3. Reuse elements from session with A
4. From the previous session
5. Decrypt with its private key
6. Verify A’s signature
7. Recover key $K$

Bob can impersonate Alice with Carol!
Denning-Sacco Fix

1. A, B
2. $S_T(B, \text{pk}_B), S_T(A, \text{pk}_A)$
3. Generate timestamp $T_A$
4. Generate random $K$
5. $E(\text{pk}_B, S_A(A, B, K, T_A)), S_T(B, \text{pk}_B), S_T(A, \text{pk}_A)$

Add the names of the parties

6. Decrypt with its private key
   Verify A’s signature
7. Verify names A and B are in message
   Recover key $K$

Cannot be re-used with Carol!

CEN-5079: 2.October.2017
Denning-Sacco Lessons

- Better be prudent than efficient
- Include more rather than less information
- Timestamps, random nonces, names of participants
Woo-Lam

1. \( A, B \)

2. \( S_T(B, \text{pk}_B) \)

3. Generate random \( R_A \)

4. \( E(\text{pk}_B, A, R_A) \)

5. \( E(\text{pk}_A, S_T(R_A, K, A, B), R_B) \)

6. \( A, B, E(\text{pk}_T, R_A) \)

7. Generate random \( K \)

8. \( S_T(A, \text{pk}_A), E(\text{pk}_B, S_T(R_A, K, A, B)) \)

9. Verify T’s signatures

10. Generate random \( R_B \)

11. \( A, E(\text{pk}_B, A, R_A) \)

12. Verify T’s signature

13. Verify \( R_A \)

14. \( E_K(R_B) \)

Verify \( R_A \)