Module 10: Authentication

Disclaimer: parts from Stefan Katzenbeisser, Günter Schäfer

Dresden, WS 14/15
„Who’s Brian of Nazareth? We have an order for him to be released.“
Identifying humans and Entity Authentication

Goal: Identify a subject (user or process!) and verify identity

Classes of authentication:
• User authentication (login)
• Computer network authentication (identity management)
• Identity verification service

Authentication cardinality:
• One-way authentication
  • Computer authenticates user
  • ATM authenticates cardholder
  • Browser authenticates Web server
• Two-way (mutual) authentication
  • ePass <--> reader
  • UMTS cellphone < -- > network
  • Online bank < -- > account holder (w/ certificates)
Different factors can be used to authenticate a user

- **Knowledge factors**
  - Passwords
  - Answers to „security questions“
  - …

- **Possession factors**
  - Security token
  - Smart card
  - Keys/certificates
  - …

- **Inherence factors**
  - Biometric factors
  - Signature
  - …

- **Sometimes: other properties (e.g. location)**
Entity Authentication

Factor verification:
- **direct** (Alice vs. Bob) or
- mediated by an **arbiter** („TTP“, Kerberos, Shibboleth)

Basic requirements:
- Strength of secret determined by its entropy (passwords, biometry)
- Provision and management: factors must remain secret (impersonation), be adjustable, possibility for revocation
- Monitoring, detection and reaction of/to malicious authentication attempts

Multi-factor authentication:
- Combines different factors (examples?)
  - ATM card (possession) and PIN (knowledge)
  - Password (knowledge) and mobileTAN (possession of cell phone)
- Requires independence of factors
- Increases security only as much as weakest factor (security question?)
- *(not to confuse with fall-back authentication – as secure as weakest factor...)*
Trouble with Authentication

Verification of factors over networks is difficult, possible with crypto

Entity authentication is more than exchange of authentic messages:

• Even if Bob receives authentic messages from Alice during a communication, he can not be sure, if:
  – Alice is actually participating in this specific moment, or if
  – Eve is replaying old messages from Alice

• Especially important, when authentication is only performed at connection-setup time:
  – Example: transmission of a (possibly encrypted) PIN when logging in

• Two principle means to ensure timeliness in cryptographic protocols:
  – Timestamps (require loosely synchronized clocks)
  – Random numbers/Nonces (challenge-response exchanges)
Authentication „in RL“

- Identity cards

User authentication

- Knowledge-based: passwords
- Inherence-based: Biometry

Network authentication:

- Kerberos
Consider how you authenticate „in RL“ (@home, bank,…)

What one *is* (inheritence)
- hand geometry
- fingerprint
- picture
- hand-written signature
- retina-pattern
- voice
- typing characteristics

What one *has* (possession)
- paper document
- metal key
- magnetic-strip card
- smart card (chip card)
- calculator

What one *knows* (knowledge)
- password, passphrase
- answers to questions
- calculation results for numbers

eID-card
Knowledge-based authentication: passwords

Simple approach

User → Server

```
Login  | Password
-----  |--------
...    | ...    
dog    | bone   
...    | ...    
```
Password based authentication

Simple approach – **security problems**

User → (dog, bone) → Server

<table>
<thead>
<tr>
<th>Login</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>dog</td>
<td>bone</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Attacker might get access!
Password based authentication

Enhanced approach using one way (hash) functions

User

Server

(dog,bone)

Login | Password
--- | ---
... | ...
dog | h(bone)
... | ...

Calculated hash:

\[ h(\text{bone}) \]

Yes: grant access

No: deny access
Password based authentication

Enhanced approach using one way (hash) functions

User

Server

(calculated)

$h$(bone)

Login | Password
--- | ---
... | ...
dog | $h$(bone)
... | ...

Slightly reduced risk, if attacker gets access.
Problems of hashed password storage

Possible attack:

• pre-computation (rainbow tables)

countermeasures:

• Hash rounds
  – store: $h^{1000}(pw)$
  – linear overhead per string (computation, storage)

• Salting
  – Use (long) random value
  – store: $h(salt,pw)$, salt

\[ \text{Login} \quad | \quad \text{Password} \]

$... \quad | \quad ...

dog \quad | \quad h(bone)

... \quad | \quad ...

\$ sudo less /etc/shadow | grep strufe
strufe:$6$m40rV3LS$kAf.4WUEwr7[...]

• $\Rightarrow$ pre-computation has to be done for each possible salt, $|\text{rounds}|$
Problems of hashed password storage

Remaining attack:

- dictionary attack
- problem: people do not chose passwords randomly
- often names, words or predictable numbers, recently personal information are used
- attacker uses dictionaries and social networks for brute force attack
- prominent program: John the Ripper
  - supports dictionary attacks and password patterns

Possible solutions:

- enforce password rules
  - consider usability
- pre-check (monitor) passwords (e.g. using John)
- train people to “generate” good passwords
  - Example: sentence → password
  - “This is the password I use for Google mail” → “Titplu4Gm”
Knowledge-based: Challenge - Response

Prerequisite: Both parties agree on common secret key

Here: Bob wants to authenticate Alice

Alice:
Key $K_{ID}$, Identification ID
Send Login-Information

Bob
Key $K_{ID}$ for ID
Generate RAND (Challenge)

(1) ID
(2) RAND
(3) C

$E(RAND, K_{ID}) = C$

Test: $C' = C$?
Assuming the Dolev-Yao adversary:

• Plaintext space for Challenges RAND has to be large!
  • Mallory can intercept and store all tuples (RAND,C)
  • She hence can replay old response

• Cipher has to be known plaintext secure

• Man-in-the-middle attacks, Replay attacks?
Identifying an individual by physiological characteristics

Requirements:

- **Universality**: Applies to all individuals
- **Uniqueness**: Different for all individuals
- **Stable**: Property does not change over lifetime
- **Measurable**: Reliably, easily with (cheap) sensors
- **Acceptance**: By users
- Resistant to **forgery**
Inherence factors: Biometry

Two-stage process:

- Enrollment: Registration of users
- Verification: Capture and compare biometric data with samples stored during enrollment

Verification is always noisy (statistical hypothesis tests)

- False positives: unauthorized individual is authenticated (FAR) → Security at risk!
- False negatives: authorized individual is rejected (FRR) → Problems for acceptance and usability
- FAR & FRR can be adapted by choice of features/characteristics

**Advantages:**

- (+) Cannot be lost/forgotten
- (+) Hard to copy

**Disadvantages:**

- (-) Very hard to change/reset
- (-) Error rate
Inherence factors: Biometry
Inherence factors: Biometry

- Storage of **minutiae**: Branches, endpoints, etc.
- Pattern (coordinates and angle) of minutiae considered to be unique
- Compare measured minutiae with samples stored at enrollment
- Problem: Some minutiae may be missing (noisy, incomplete sensing)!

- Main biometric factor (Face recognition, retina scans increasingly used)
Arbited Authentication, Single Sign On

Goals of Single Sign On-Concepts:

- User is authenticated only once (centrally),
- no separate authentication upon requests to use different services within an administrative domain

Kerberos (MIT, 1980s):

- SSO for services within a „realm“
- Authenticate subjects („principals“): users, computers, server
- Exchange session keys for principals based on Needham-Schroeder
- Underlying cryptographic primitive of symmetric encryption (DES in Kerberos V. 4, from V. 5 on other algorithms allowed)
Objectives of Kerberos:
- **Security**: prevent impersonation of users when accessing a service
- **Reliability**: service use requires authentication --> reliability and availability
- **Transparency**: authentication beyond password transparent to user
- **Scalability**: the system has to support a large number of clients and servers

Design of Kerberos:
- A **single trusted server** per domain (Key distribution center, KDC)
- Tasks of the KDC are:
  - Authenticating the clients of its domain
  - Issue tickets as authentication tokens

Authentication of a Principal
- **Idea: Pre-Shared Secrets** between Principal and KDC
  - For users: hashed (MD5) passwords, master key $K_A$ is derived
  - For servers: shared, secret master key $K_S$
Content of a Ticket:

- Each ticket is valid only for a principal \( C \) (e.g. Joe) on the specific server \( S \) (e.g. NFS), for a specific time:

- \( T_{c,s} = S, C, addr, timestamp, lifetime, K_{c,s} \) with:
  - \( S \): Name of the Server,
  - \( C \): Name of the requesting client,
  - \( addr \): IP address of the requesting client,
  - \( timestamp \): current time,
  - \( lifetime \): lifetime of the ticket,
  - \( K_{c,s} \): Session key for the communication between \( S \) and \( C \)
Kerberos (Protocol, simplified)

- User Joe logs in to local PC with a password

- Local PC (client) C sends ID and Nonce to KDC and requests a ticket for the TGS:
  
  Joe → KDC: Joe, TGS, *Nonce1*

- KDC extracts master key of the user from its database and issues a ticket $T_{Joe,TGS}$ to authorize utilization of TGS
  
  KDC → Joe: $\{K_{Joe,TGS},\text{Nonce1}\}_{K_{Joe}}$, $\{T_{Joe,TGS}\}_{KTGS}$

- Client requests Joe to enter Kerberos password, derives $K_{Joe}$ and extracts: $K_{Joe,TGS}$ *Nonce1*
Kerberos (Protocol, simplified, ctd.)

- Client requests ticket at TGS to use the NFS server:
  
  Joe → TGS: \( \{ A_{joe} \} \ K_{Joe,TGS}, \{ T_{Joe,TGS} \} \ K_{TGS}, \ NFS, \ Nonce2 \)

  where \( A_{joe} = Joe, IP-Addr, timestamp \) is called „Authenticator“

- TGS checks Authenticator and sends a ticket for the NFS server to Joe:

  TGS → Client: \( \{ K_{Joe,NFS}, Nonce2 \} \ K_{Joe,TGS}, \{ T_{Joe,NFS} \} \ K_{NFS} \)

- Joe uses ticket at NFS server:

  Client → NFS: \( \{ A_{joe} \} \ K_{joe,NFS}, \{ T_{joe,NFS} \} \ K_{NFS} \)

- For mutual authentication:

  NFS → Client: \( \{ timestamp+1 \} \ K_{joe,NFS} \)
Kerberos

1. Request TGT
2. TGT, Session Key
3. Request SGT
4. SGT, Session Key
5. Request Service
6. Service Authenticator

Authentication Server
(user data)

Ticket Granting Server
(access data)

NFS

DB

Printer
...

Privacy and Security

Inter-Realm Authentication

Extending Kerberos to multiple realms:

- Establish mutual trust between TGS of different realms: $K_{TGS1,TGS2}$

1. Request TGT
2. TGT, Session Key
3. Request TGT$_{rem}$
4. TGT$_{rem}$, Session Key
5. Request SGT
6. SGT, Session Key
7. Request Service Authentication
8. Service Authentication

Realm 1

Realm 2
Summary

You know means for confidentiality & integrity other than crypto ;-) 

You can distinguish between authorization, authentication, access control 

You can explain ACMs, ACLs and capabilities 

You can distinguish DAC and MAC 

You know the different general strategies of IBAC and RBAC 

You can explain different classes of authentication and factors 

You know about cardinality and multi-factor authentication 

You can explain some knowledge-based authentication schemes, strategies 

You about biometry and its advantages and disadvantages 

04.02.2015