Authentication methodologies

- can be based on different factors (1/2/3-factors authentication):
  - something I know (e.g. a password)
  - something I have (e.g. magnetic card)
  - something I am (e.g. my fingerprint)

- multiple different mechanisms can be combined to achieve identification

User authentication

server
authentication request
uid : f(S UID)
uid

user (uid)
proof request
proof = F(S UID)
secret (S UID)
Password-based authentication

- secret = the user password
- (client) create and transmit proof
  - \( F = I \) (the identity function)
  - i.e. proof = password (cleartext!)
- (server) verify the proof:
  - case #1: \( f = I \) (the identity function)
    - server knows all passwords in cleartext (!)
    - access control: proof = password ?
  - case #2: \( f = \) one-way hash (that is a digest)
    - server knows the passwords' digests, \( H_{UID} \)
    - access control: \( f(\text{proof}) = H_{UID} ? \)

Password (reusable)

User

<table>
<thead>
<tr>
<th>User (UID)</th>
<th>Authentication request</th>
<th>UID</th>
<th>Password request</th>
<th>Secret (PUID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UID : ( f(\text{PUID}) )</td>
<td></td>
<td></td>
<td>PUID</td>
<td></td>
</tr>
</tbody>
</table>

Password-based authentication

- pro:
  - simple for the user
  - ... if she has to remember just one password
- cons:
  - user-side password storage (post-it!)
  - password guessable (my son's name!)
  - password readable during transmission
  - server-side password storage – the server must know in cleartext the password or its unprotected digest (dictionary attack)
Password

- suggestions to reduce the associated risks:
  - alphabetic characters (uppercase + lowercase) + digits + special characters
  - long (at least 8 characters)
  - never use dictionary words
  - frequently changed (but not too frequently!)
  - don’t use them :-)
  - use of at least one password (or PIN or access code di or ...) unavoidable unless we use biometric techniques

Storing the password

- NEVER in cleartext!
- encrypted password? then the server must know the key in cleartext …
- store a digest of the password
- … but beware of the dictionary attack:
  - let HP be the hash of a password
  - for (each Word in Dictionary) do
    - if ( hash(Word) = HP) then write (“found!”, Word)
  - may be made faster by using a “rainbow table”
- we must therefore insert an unexpected variation, usually named “salt”

Using the salt in storing passwords

- for each user UID:
  - create / ask the pwd
  - generate a random salt (should contain rarely used or control characters)
  - compute HP = hash ( pwd || salt )
  - store the triples { UID, HP, saltUID }
  - additionally we have different HP for users having the same pwd
  - makes the dictionary attack nearly impossible (included those based on rainbow tables)
(Symmetric) challenge-response systems

- A challenge (typically a random number) is sent to the user ...
- ... who replies with the solution after a computation involving the shared secret and the challenge
- The server must know the secret in clear
- Often R is a hash function (can't be encryption)

\[
\text{response } = R(\text{challenge}, \text{SUID})
\]

Mutual authentication with symmetric challenge (v1)

Mutual authentication with symmetric challenge (v2)
(Asymmetric) challenge-response systems

- A random number R is encrypted with the user’s public key ...
- ... and the users replies by sending R in clear thanks to its knowledge of the private key

\[
\text{challenge} = E(R, K_{pub\text{Lioy}}) \\
\text{response} = R
\]

Acceptable users

Risks with asymmetric challenges

- Trust in the issuer CA of the user cert
- Check of the name constraint on trusted CAs
- Unwilling RSA signature possible:
  - If \( R = \text{digest(document)} \) ...
  - And the server sends R in clear and ask it back encrypted with user’s private key ...
  - Then the user has unwillingly signed the document!!!

Password (one-time)

\[
P_48 = p(48, S_{UID})
\]
One-Time Passwords (OTP)

- original idea:
  - Bell Labs
  - the S/KEY system
  - public-domain implementation
- commercial implementations with automatic hardware generators (authenticator)

OTP provisioning to the users

- on "stupid" or insecure workstation:
  - paper sheet of pre-computed passwords
  - hardware authenticator (crypto token)
- on intelligent and secure workstation:
  - automatically computed by an ad-hoc application
  - eventual integration into the communication sw (e.g. telnet client) or hw (e.g. modem)

The S/KEY system (I)

- RFC-1760
- the user generates a secret S (the seed)
- the user computes N one-time passwords:
  - $P_1 = h(S)$
  - $P_2 = h(P_1) = h(h(S))$
  - ...
- the user initializes the authentication server with the last generated password (e.g. $P_{100}$)
The S/KEY system (II)
- The server prompts for the passwords in reverse sequence:
  - S: P99?
  - C: X
  - S: if \( h(X) = P100 \) then access allowed + X is stored
- In this way the server doesn't need to know the client’s secret
- RFC-1760 uses MD4 (other choices possible)
- Public-domain implementation for Unix, MS-DOS, Windows, MacOS

S/KEY – password generation
- The user inserts a pass phrase (PP) at least 8 characters long
- The PP is concatenated with a seed provided by the server
- A 64 bit quantity is extracted from the MD4 hash (by XORing the first/third 32 bit groups and the second/fourth groups)

S/KEY – passwords
- 64 bit passwords are a compromise
- Neither too long nor too short
- Possible typing as a sequence of 6 short English words chosen from a dictionary of 2048
  (e.g. “A”, “ABE”, “ACE”, “ACT”, “AD”, “ADA”)
- Client and server must share the same dictionary
OTP problems

- generally uncomfortable
- uncomfortable when used to access multiple password-based services (e.g. POP with periodic check of the mailbox)
- expensive when based on hw authenticators
- paper-based passwords cannot be used by a process but only by a human operator

Problems of hw authenticators

- denial-of-service:
  - deliberately wrong attempts to trigger account blocking
- social engineering:
  - phone call to simulate loss of the authenticator and remotely initialize a new one

Password (one-time token-based)

\[ P_{1107, UID} = p (11:07, S_{UID}) \]
RSA SecurID

- invented and patented by Security Dynamics
- time-based synchronous OTP technique:
  \[ P_{UID}(t) = h(S_{UID}, t) \]
- access code (token-code):
  - 8 digits
  - random, never repeats itself
  - changes every 60 s
  - maximum drift 15 s/year
  - expires in 4 years
- based on proprietary and secret (!) hash algorithm

SecurID: architecture

- the client sends in clear: user, PIN, token-code (seed, time)
- based on user and PIN the server verifies against three possible token-codes: TC1, TC2, TC3
- duress code: PIN to generate an alarm (useful for authentication under threat)
- wrong authentication attempts limited (default: 10)
- may have three different keys per device

SecurID: hardware

- SecurID Card: classic device (credit-card size)
- SecurID PinPad: local PIN keying and then only user and token-code* are sent to the server
- SecurID Key Fob: usable as a key fob
- SecurID modem: PCMCIA-II V.34 modem with an internal token activated via sw by introducing the PIN
**SecurID: software**

- **SoftID**
  - works as a SecurID PinPad but is a sw application
  - automatic or manual transmission of the token-code
  - problem: clock synchronization

**SecurID: architecture**

**SecurID: client**

- **ACE/client**
  - manages the dialogue with the ACE/server
  - encrypted channel
  - `sd_ftp` for secure FTP
- **available for:**
  - Unix
  - Win32
  - Netware
  - Macintosh
  - TACACS
SecurID: server

- ACE/server:
  - authentication with SecurID tokens
  - monitor, audit and report
  - GUI management interface
  - authentication API
  - SQL interface to access a DBMS (already) storing the user data
  - large commercial support in security (e.g. firewall) and communication (e.g. comm. server) products
  - available for Solaris, AIX, HP-UX, NT, 2000, XP

CRYPTOCARD

- challenge-response mechanism
- based on DES-CBC
- single product: RB-1 card
- 8 digits (hex, dec) LCD display
- user-replaceable battery (change every 3-4 years)
- to avoid inserting the challenge, can store the last one and automatically compute the next one
- server for Unix and Windows (Radius, Tacacs+)

CRYPTOCARD: hardware
**Authentication of human beings**
- how can we be sure of interacting with a human being rather than with a program (e.g. sensing a password stored in a file)?
- two solutions:
  - CAPTCHA techniques (Completely Automated Public Turing test to tell Computers and Humans Apart)
    - e.g. picture with images of distorted characters
  - biometric techniques
    - e.g. fingerprint

**Biometric systems**
- measure of one biologic characteristics of the user
- main characteristics being used:
  - fingerprint
  - voice
  - retinal scan
  - iris scan
- useful to *locally* replace a PIN or a password

**Problems of biometric systems**
- FAR = False Acceptance Rate
- FRR = False Rejection Rate
- FAR and FRR may be partly tuned but they heavily depend on the cost of the device
- variable biological characteristics:
  - finger wound
  - voice altered due to emotion
  - retinal blood pattern altered due to alcohol or drug
Problems of biometric systems

- psychological acceptance:
  - “Big Brother” syndrome (=personal data collection)
  - some technologies are intrusive and could harm
- lack of a standard API / SPI:
  - high development costs
  - heavy dependence on single/few vendors
- API / SPI being developed:
  - standard and unified
  - based on CDSA

API? SPI? middleware!

<table>
<thead>
<tr>
<th>API (Application Programming Interface)</th>
<th>middleware (e.g. CDSA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>APP1</td>
<td>APP2</td>
</tr>
<tr>
<td>device / service no. 1</td>
<td>device / service no. 2</td>
</tr>
</tbody>
</table>
Kerberos
- authentication system based on a TTP (Trusted Third Party)
- invented as part of the MIT project Athena
- user password never transmitted but only used locally as cryptographic (symmetric) key
- realm = Kerberos domain, that is the set of systems that use Kerberos as authentication system
- credential = user.instance@realm

Kerberos
- ticket
  - data structure to authenticate a client to a server
  - variable lifetime
    - (V4: max 21 hours = 5’ x 255)
    - (V5: unlimited)
  - encrypted with the DES key of the target server
  - bound to the IP address of the client
  - bound to just one credential
  - simple or mutual authentication

Kerberos high-level view

© Antonio Lioy - Politecnico di Torino (1995-2009)
Kerberos: data formats (v4)

TICKET

- server-id
- client-id
- client-address
- timestamp
- life
- \( K_{B,C} \)

AUTHENTICATOR

- client-id
- client-address
- timestamp
- \( K_{B,C} \)

TGT request

- Client (C, TGS)
- Authentication Server (A, S)
- \{ \( K_{C,TGS} \), \{ \( T_{C,TGS} \) \( K_{TGS} \) \} \( K_{C} \) \}

Ticket request

- Client (C)
- Ticket Granting Server (TGS)
- \( \{ \{ T_{C,B} \} \( K_{S} \), \( K_{C,S} \) \} \( K_{C,TGS} \) \)

© Antonio Lioy - Politecnico di Torino (1995-2009)
Ticket use

Client (C) sends to the server (S):
- \( \{ TC, S \} KS \)
- \( \{ AC \} KS \)
- \( \{ \text{timestamp}(AC) + 1 \} KS \)

Kerberos versions

- MIT V4 (the original public one)
- MIT V5 (RFC-1510)
  - not only DES
  - extended ticket lifetime (begin-end)
  - inter-realm authentication
  - forwardable ticket
  - extendable ticket
- OSF-DCE
  - based on MIT V5
  - implemented as RPC rather than a message exchange protocol

Kerberos: problems

- clock synchronization required:
  - within a LAN it’s useful anyway
  - in WAN may originate problems
  - Kryptoknight (alias IBM NetSP) doesn’t require clock synchronization
- remote access needs cleartext password:
  - encrypted channel or integration with OTP, symmetric or asymmetric challenge
  - Kerberized dial-up modems
**Kerberos: advantages**

- Single login to all Kerberized services
  - K-POP, K-NFS, K-LPD
  - K-telnet, K-ftp
  - K-dbms
- The ticket mechanism is ideally for intermittent connections
  - Mobile computers
  - ISDN, WiFi
- Increasing commercial support
  (MS has adopted Kerberos since Windows-2000)

**SSO (Single Sign-On)**

- The user has a single "credential" to authenticate himself and access any service in the system
- Fictitious SSO:
  - Client for automatic password synchronization / management (alias "password wallet")
  - Specific for some applications only
- Integral SSO:
  - Multiapplication authentication techniques (e.g. asymmetric challenge, Kerberos)
  - Likely requires a change in the applications

**Interoperability**

- OATH (www.openauthentication.org)
- Interoperability of authentication systems based on OTP, symmetric or asymmetric challenge
- Development of standards for the client-server protocol and the data format on the client (draft RFC):
  - HOTP (HMAC OTP, RFC-4226)
  - Challenge-response protocol
  - Bulk provisioning
  - Symmetric key container