Kerberos, SSL, IPsec

Shankar

May 28, 2013
Kerberos 4
Kerberos 4

Authentication in network (Realm)
• Realm has KDC and users (principals)
• Users: human (log in to workstations)
  apps: NFS, rsh, etc
• Authentication: based on Needham-Schroeder protocol
• Attacker: can read and write messages in transit.
• Assumes DES and IPv4
• Uses timestamps: nodes need synchronized clocks

KDC has
• **Master key** for each user
  • weak key for human; strong key for apps
• Secret-key $K_{KDC}$ not shared with any user
  • to encrypt database (master keys), TGTs
• Database: changes only when user’s master key changes
  • mitigates KDC from becoming bottleneck
Kerberos 4

When a human user logs in
- KDC authenticates user based on user’s master key
- KDC gives user credentials encrypted with user’s master key
  - **Session key**: for current login session
    // user’s master key not used after login
  - **Ticket Granting Ticket (TGT)** encrypted by $K_{KDC}$:
    - for user to obtain further tickets from KDC

For human user to access an app
- User sends KDC $\text{enc}([\text{request, TGT, timestamp}], \text{session key})$
- KDC returns credentials encrypted with session key
  - session key to talk to app
  - ticket encrypted with app’s master key (app is not human)
- user sends app $[\text{request, ticket}]$
### K4: Login handshake

<table>
<thead>
<tr>
<th>User A (has pw) at workstation</th>
<th>KDC (has A: Kₐ)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>start login</strong></td>
<td>receive msg, retrieve Kₐ</td>
</tr>
<tr>
<td><strong>send [“A needs TGT”]</strong></td>
<td>generate session key Sₐ</td>
</tr>
<tr>
<td></td>
<td>tgtₐ ← enc([A, Sₐ], Kₐ)&lt;br&gt;crdₐ ← enc([Sₐ, tgtₐ], Kₐ)&lt;br&gt;send [crdₐ ]</td>
</tr>
<tr>
<td>receive msg</td>
<td></td>
</tr>
<tr>
<td>get Kₐ from pw</td>
<td></td>
</tr>
<tr>
<td>extract Sₐ, tgtₐ from crdₐ</td>
<td></td>
</tr>
<tr>
<td>forget pw;</td>
<td></td>
</tr>
<tr>
<td>// use Sₐ henceforth</td>
<td></td>
</tr>
<tr>
<td>finish login</td>
<td></td>
</tr>
</tbody>
</table>
# K4: Accessing remote app B
*(LATER IN THE SESSION)*

<table>
<thead>
<tr>
<th>user A at workstation</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>rlogin B</td>
<td>receive msg from A</td>
</tr>
<tr>
<td>send ([A, B, \text{tgt}_A, \text{enc}(ts, S_A)]) // (S_A(ts)): authenticator</td>
<td>extract K, ts</td>
</tr>
<tr>
<td></td>
<td>send ([B, A, \text{enc}(ts+1, K)])</td>
</tr>
<tr>
<td>rcv msg, gen sess key K</td>
<td>receive msg</td>
</tr>
<tr>
<td>get (S_A) from (\text{tgt}_A)</td>
<td>extract K, (\text{enc}(ts, K))</td>
</tr>
<tr>
<td>get ts and verify</td>
<td>receive msg</td>
</tr>
<tr>
<td>find B’s master key (K_B)</td>
<td>end</td>
</tr>
<tr>
<td>(\text{tkt}_B \leftarrow \text{enc}([A, K], K_B))</td>
<td></td>
</tr>
<tr>
<td>(\text{crd}_B = \text{enc}([B, K, \text{tkt}_B], S_A)) // credential</td>
<td></td>
</tr>
<tr>
<td>send ([\text{crd}_B]) to A</td>
<td></td>
</tr>
</tbody>
</table>
K4: Replicated KDCs for performance

- One master KDC and several secondary KDCs
- Each secondary KDC has read-only copy of KDC database
- Additions/deletions/changes to master keys always done at master KDC
- Secondary KDCs can generate session keys, TGTs, etc.
- Master disseminates KDC database to secondary KDCs with integrity protection (master keys already encrypted with $K_{KDC}$)
# K4: Authentication across multiple realms

- Possible only if their KDCs share a key.
- Principal id = [name, instance, realm], each 40 chars max

<table>
<thead>
<tr>
<th>A in realm X</th>
<th>KDC X</th>
<th>KDC Y</th>
<th>B in realm Y</th>
</tr>
</thead>
</table>
| send 

\[
([A.X, B.Y])
\]

to X

receive msg

send 

\[
([\text{cred}.Y])
\]

to A

receive msg

send 

\[
([A.X, B.Y, \text{cred}.Y])
\]

to Y

receive msg

send 

\[
([A, B, \text{cred}.B])
\]

receive msg
K4: Key version number

- If A has ticket to B, and B changes its master key, then ticket no longer valid.
- To handle this (w/o A asking KDC for a new ticket):
  - Apps remember old master keys up to expiry time (approx 21 hrs)
  - In tickets, the key is sent along with version number
  - Human users need not remember old passwords
K4: Network layer address in tickets

- Ticket has IPv4 address of the user given the ticket
- Received ticket is not accepted if ticket sender’s IP address does not match
- So if B is to impersonate A, it must also spoof the IP address of A (easy to do)
- Prevents delegation
  - A cannot have B at another IP address do work on behalf of A (unless B spoofs IP address of A !)
K4: Encryption/Integrity of data

- After authentication, data exchange can be any of
  - clear
  - encrypted
  - integrity-protected
  - encrypted and integrity-protected
- Choice is up to the application (performance vs security)
- K4 uses adhoc integrity protection (not so safe)
Kerberos 5
Kerberos 5

- More general than Kerberos 4

- Message formats defined using ASN.1 and BER
  - So allows for addresses of different formats, etc.
  - Occupies more octets

- Names: [NAME, REALM]
  - Arbitrary content, length // allows “.”, “@”, ...
  - Allows X.500 names // country/org/name/...

- Allows choice of crypto algorithms
  - Uses proper integrity protection
K5: Delegation of Rights

- User A can ask KDC for a TGT with
  - network addresses different from A’s address
    (for use by a principal at another address on behalf of A)
  - no address (for use by any principal at any address)
- User A can give a tgt/tkt to B with specific constraints
  - TGT/tkt has “app” field copied by KDC to any derived tkt
- A’s TGT can be **forwardable**:
  - A can use it to get a TGT (for B) with different address.
  - A also says whether derived TGT is itself forwardable
- A’s TGT can be **proxiable**:
  - A can use it to get tkt (for B) with different address
- Ticket lifetime
K5: TGT/TKT Lifetime

- Fields:
  - start-time: when ticket becomes valid
  - end-time: when ticket expires (but can be renewed)
  - auth-time: when A first logged in (from initial login TGT)
  - renew-till: latest time for ticket to be renewed.

- Allows unlimited duration subject to renewing (e.g., daily)
  - exchange tgt/tkt at KDC for a new (renewed) tgt/tkt
  - tgt/tkt has to be renewed before expiry

- Allows **postdated** tickets (e.g., for batch jobs).
K5: Keys

- KDC remembers old master keys of human users also
  - because tgts/tickets are renewable and can be postdated.
- For each principal B, KDC stores
  - key: B’s master key encrypted with $K_{KDC}$ (current or past)
  - p_kvno: version number of B’s master key
  - k_kvno: version number of $K_{KDC}$ used to encrypt
  - max_life, max_renewable_life: for tickets issued to B
  - expiration: when this entry expires
  - mod_date/mod_name: when entry last modified, by who
  - flags: preauthentication?, forwardable?, proxiable?, …
  - …
- Human user master key derived from pw and realm name.
  - So weak protection from key exposure across realms
K5: Authentication Chains

- Allows KDC chains of authentication (unlike V4)
- Example: KDCs A, B, C, where
  - A-B share key, B-C share key, but A,C do not.
  - K5 allows C to accept tkt sent by A and generated by B
- Each ticket includes all the intermediate KDCs
  - receiving KDC can reject tkt if it has suspect intermediary
K5: Evading off-line password guessing

- K4 allows off-line password guessing:
  - KDC does not authenticate login request before issuing TGT
  - So B can spoof A, get a TGT for A, do off-line dictionary attack on TGT

- In K5
  - Login req must contain $K_A\{\text{timestamp}\}$; so above attack not possible
  - KDC also does not honor requests for tickets to human users by others
    - Prevents logged-in B asking KDC for a tkt (to delegate) to A, on which it can do off-line password guessing.
K5: Key inside authenticator

- Suppose A and B share a session key K generated by KDC.
- A and B can have another (simultaneous) session using a different key.
- This can be done without involving the KDC:
  - A makes up a key for this second session and gives that to B encrypted by K.
K5: Double TGT Authentication

- Allows A to access server B that has session key, say $S_B$, but not master key $K_B$
- Needed for X windows:
  - X server manages display on workstation screen
  - X clients (e.g., xterm) run on local or remote workstations
  - X client (A) needs tkt to X server (B) to display on screen
- No good for A to get from KDC a (regular) tkt encrypted $K_B$
- Instead
  - A gets TGT$_B$ from B
    - sends [“A to B”, TGT$_A$, TGT$_B$] to KDC
  - KDC
    - extracts $S_B$ from TGT$_B$ (encrypted with $K_{KDC}$)
    - creates session key $K_{AB}$,
    - generates tkt$_B$ encrypted with $S_B$ (i.e., enc([‘A’, $K_{AB}$], $S_B$) and sends to A
K5: X windows

<table>
<thead>
<tr>
<th>B at workstation/X server</th>
<th>C (may be B’s workstation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>login to X server (B,pw)</td>
<td></td>
</tr>
<tr>
<td>• request TGT&lt;sub&gt;B&lt;/sub&gt; from KDC</td>
<td></td>
</tr>
<tr>
<td>• obtain [S&lt;sub&gt;B&lt;/sub&gt;, TGT&lt;sub&gt;B&lt;/sub&gt;] from KDC</td>
<td></td>
</tr>
<tr>
<td>• forget B’s passwd</td>
<td></td>
</tr>
<tr>
<td>• start serving B (eg, keybd, mouse)</td>
<td></td>
</tr>
</tbody>
</table>

request X client at C (eg, xterm)

- X client starts
- has info to display at B’s screen
- get TGT<sub>B</sub> from X server
- ask KDC for tkt encrypted by S<sub>B</sub>
- present tkt to X server and info to display

- X server displays client’s info
Security with TCP/IP
TCP/IP + Security

- TCP/IP stack without security

<table>
<thead>
<tr>
<th>apps</th>
<th>TCP</th>
<th>UDP</th>
<th>...</th>
<th>apps</th>
<th>TCP</th>
<th>UDP</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IP</td>
<td>LRD channel</td>
<td></td>
<td>IP</td>
<td>LRD channel</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- TCP provides apps with
  - connection establishment
  - reliable data transfer

- Want to extend this to handle attackers
  - network attackers: passive / active
  - endpoint attackers: send messages with arbitrary fields
  - **authentication**: extends connection establishment
  - confidentiality, **integrity**: extends reliable data transfer
Natural solution: Secure TCP

<table>
<thead>
<tr>
<th>apps</th>
<th>TCP</th>
<th>UDP</th>
<th>Secure TCP</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UDP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secure TCP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LRD/attacker channel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>apps</td>
<td>TCP</td>
<td>UDP</td>
<td>Secure TCP</td>
<td>...</td>
</tr>
<tr>
<td>TCP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UDP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secure TCP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

STCP (Secure TCP) like TCP except
- 3-way connection establishment includes
  - client id, server id, authentication secret
- data transfer is tcp-like except
  - IP header is in clear
  - secure-tcp header encrypted
  - secure-tcp payload encrypted
Reality

Implementors did not want

• modifications to TCP (which is in OS kernel)
• another protocol like TCP in OS kernel or over UDP
• another protocol like TCP in app space (eg, over UDP)

Instead we now have two partial solutions

• SSL (Secure Sockets Layer): above TCP
• IPsec: above IP and below transport layer (TCP, UDP)
SSL
SSL

- When A connects to B
  - TCP A and TCP B establish a connection
  - SSL A and SSL B authenticate over TCP
    - using A public key and B public key, or
    - using B public key and A password (typical)
- During data transfer:
  - SSL encrypts outgoing / decrypts incoming
  - TCP messages have TCP header in clear
  - Easy DOS attack: Rogue packet attack
SSL

client A  ssl A  tcp A  tcp B  ssl B  server B

tcp conn est handshake

auth handshake using K establish session key(s)

plain text  disconnect  plain text
SSL

- SSL A: generate \( R_A \)
  send \([B, \text{ciphers supported}, \text{random } R_A]\]
- SSL B: choose cipher, generates \( R_B \)
  send \([B, \text{cipher chosen}, \text{cert}_B, \text{random } R_B]\]
- SSL A: generate \( S \) \hspace{1cm} \text{// pre-master secret}
  \( K \leftarrow f(S, R_A, R_B) \) \hspace{1cm} \text{// master secret}
  send \([\text{enc}_P(S, \text{pub}_B), \text{hash}_1(\text{handshake}, K)]\]
- SSL B: send \([\text{hash}_2(\text{handshake})]\]
- SSL A: if \( \text{hash}_2 \) verifies, B authenticated
  send \([\text{enc}(\text{pw}, K\text{-derived-key})]\]
- SSL B: if \( \text{pw} \) verifies, A authenticated

A can also use \( \text{cert}_A \) for authenticating itself to B
SSL

- S: pre-master secret
- K: master secret
  - $K = f(S, R_A, R_B)$
- Keys for data encryption/integrity obtained from $K, R_A, R_B$
  - A’s write (transmit) key = B’s read (receive) key
  - B’s write (transmit) key = A’s read (receive) key

- A does two public-key crypto operations
  - verifying cert$_B$
  - calculating $\{S\}_B$
- To minimize this, S can be reused across different sessions
  - motivated by http 1.0 (opens many A-B tcp sessions)
  - session id
SSL

ssl A

[B, ciphers, R_A] →

← [B, session-id = X, cert_B, cipher, R_B]

new session later on

[x,y,B, session-id = X, ciphers, R_A] →

← [B, session-id = X, cert_B, cipher, R_A, keyed hash of handshake]

[x,y, keyed hash of handshake] →

ssl A

initial session

if ssl A still has X:S, can reuse it

5/29/2013 shankar

authentication slide 31
IPsec
**IPsec**

- IPsec sits above IP and below TCP, UDP, ...
- IP packet: [IP hdr, IPsec hdr, TP hdr, TP payload]
  \[\leftarrow IP \text{ payload} \rightarrow\]
  \[\leftarrow IPsec \text{ payload} \rightarrow\]
- TP is IP: “tunnel” mode  // often used to tunnel IP traffic
- TP is not IP: “transport” mode
IPsec: generic header

- IP hdr
  - sender ip addr, rcvr ip addr
  - hop count  // mutable
  - next protocol id: TCP, UDP, IP, IPsec (AH or ESP), ...

- IPsec header (generic):
  - SPI (security parameter index): identifies IPsec connection (SA)
  - sequence number: of IPsec packet (for replay attacks)
  - IV (for encryption/integrity)
  - authentication data (integrity check)
  - next protocol id: (TCP, UDP, IP, ...)
IPsec: Security association

- IPsec SA (security association): IPsec connection
- An SA is one-way, so need two SAs for bi-directional flow

- IPsec entity in a node has
  - **Security policy database** (control path)
    - for <ip addr, port, etc>:
      - crypto or not? type, integrity/encryp, ...
  - **SA (security association) database** (data path)
    - outgoing: for remote ip addr:
      - SPI, crypto key/alg, sequence number
    - incoming: for SPI:
      - crypto key/algo, expected seq number, ...
IPsec: AH and ESP

- IPsec headers are in two flavors

  - AH hdr:
    - SPI, sequence number, auth data, next protocol id
    - integrity only but on enclosing IP
      - payload + “immutable” header
    - not compatible with NAT, firewalls

  - ESP hdr:
    - SPI, seq number, IV, auth data, next protocol id
    - integrity and/or encryption on enclosing IP payload
    - compatible with NAT, firewalls
IPsec: IKE

- For an IPsec SA to operate, its parameters (integrity/encryp, key, ...) must be set in the (SA database of the) end-points of the SA
- Can be done manually by end-point administrators or dynamically using IKE
- IKE runs over UDP, has two phases, and is an UGLY MESS