Network Security: Classic protocols and flaws, Kerberos

Tuomas Aura
T-110.5240 Network security
Aalto University, Nov-Dec 2011
Outline

- Classic key-exchange protocols and flaws
- Kerberos authentication
- Kerberos in Windows domains
Classic key-exchange protocols and flaws
Needham-Schroeder secret-key protocol

- The first secret-key key-exchange protocol 1978
- Kerberos is based on this protocol
- Trusted server T shares a secret master key with everyone. It distributes a session key encrypted with the master keys:

1. A → T: A, B, N_{A1}
2. T → A: E_{TA}(N_{A1}, B, K_{ses}, ticket_{AB})
3. A → B: ticket_{AB}, E_{ses}(N_{A2})
4. B → A: E_{ses}(N_{A2-1}, N_{B})
5. A → B: E(N_{B-1})

K_{TA}, K_{TB} = A’s and B’s master keys
K_{ses} = session key selected by T
ticket_{AB} = E_{TB}(K_{ses}, A)

“Encryption” must also protect integrity
Needham-Schroeder analysis

The protocol again:
1. $A \rightarrow T$: $A, B, N_{A1}$
2. $T \rightarrow A$: $E_{TA}(N_{A1}, B, K_{ses}, \text{ticket}_{AB})$  // $\text{ticket}_{AB} = E_{TB}(K_{ses}, A)$
3. $A \rightarrow B$: $\text{ticket}_{AB}, E_{ses}(N_{A2})$
4. $B \rightarrow A$: $E_{ses}(N_{A2}-1, N_B)$
5. $A \rightarrow B$: $E_{ses}(N_B-1)$

- $T$ encrypts session key under $A$’s and $B$’s master keys
- Authenticators in messages 4–5 for key confirmation
- $N_{A1}$ guarantees freshness of ticket and session key to $A$
- $N_{A2}$ and $N_B$ guarantee freshness of authenticators to $A$ and $B$, respectively
- But no freshness of the ticket to $B$...
Needham-Schroedder vulnerability

The protocol again:

1. $A \rightarrow T$: $A, B, N_{A0}$
2. $T \rightarrow A$: $E_{TA}(N_{A0}, B, K_{ses}, ticket_{AB})$  // $ticket_{AB} = E_{TB}(K_{ses}, A)$
3. $A \rightarrow B$: $ticket_{AB}, E_{ses}(N_A)$
4. $B \rightarrow A$: $E_{ses}(N_A-1, N_B)$
5. $A \rightarrow B$: $E_{ses}(N_B-1)$

Vulnerability discovered by Denning and Sacco 1981

- Freshness of the ticket not guaranteed to $B$
- Assume attacker compromises an old session key and has sniffed the corresponding ticket, or
- Or assume attacker compromises $A$’s master key $K_{TA}$. $A$’s master key is replaced but, before that, the attacker manages to obtain a ticket for $B$

Replay attack by $C$ who knows an old session key, pretending to be $A$:

3’. $C(A) \rightarrow B$: $ticket_{AB-old}, E_{ses-old}(N_A)$  // $ticket_{AB-old} = E_{KTB-old}(K_{ses-old}, A)$
4’. $B \rightarrow C(A)$: $E_{ses-old}(N_A-1, N_B)$
5’. $C(A) \rightarrow B$: $E_{ses-old}(N_B-1)$

Lesson: protocols should withstand compromise of old secrets

How to fix?
Denning-Sacco protocol

- Public-key key exchange 1981; flaw found in 1994
- A obtains certificates from trusted server T for both A’s and B’s public keys
  1. A → T: A, B
  2. T → A: $\text{Cert}_A, \text{Cert}_B$
  3. A → B: $E_B(T_A, K_{ses}, S_A(T_A, K_{ses}))$, $\text{Cert}_A, \text{Cert}_B$

$K_{ses} = \text{session key selected by A}$

$E_B = \text{encryption with B’s public key}$

$\text{Cert}_A = A, \text{PK}_A, S_T(A, \text{PK}_A)$

- Analysis:
  - Expiration time missing from certificates → should be added!
  - Public-key encryption for secrecy of $K_{ses} → \text{ok}$
  - Time stamp for freshness → should do something about fast replays!
  - Public-key signature for authentication → but what information exactly is authenticated?
Denning-Sacco vulnerability

The protocol again:
1. A → T: A, B
2. T → A: Cert_A, Cert_B
3. A → B: E_B(T_A, K_{ses}, S_A(T_A, K_{ses})), Cert_A, Cert_B

The signed message $S_A(T_A, K_{AB})$ does not contain all possible information

What is missing?
The signed message is not bound to the identity of B

Does it matter when only B can decrypt the message?
Yes because B could be bad!

B can forward the last message to anyone else, e.g. to C:

$$3'. B(A) \rightarrow C: E_C(T_A, K_{AB}, S_A(T_A, K_{AB})), Cert_A, Cert_C$$

C will think it shares $K_{ses}$ with A but it is really shared with B

Legitimate user B can impersonate legitimate user A.
Lesson: protocols should withstand insider attacks

How to fix?
Needham-Schroeder public-key protocol

The first public-key key-exchange protocol 1978; flaw found in 1995 [Lowe95]

A and B know each other’s public encryption keys or obtain certificates from a trusted server T as needed. Then, A and B exchange key material:

1. \( A \rightarrow B: E_B(N_A, A) \)
2. \( B \rightarrow A: E_A(N_A, N_B) \)
3. \( A \rightarrow B: E_B(N_B) \)

\( N_A, N_B \) = secret nonces, also serving as key material
\( E_A, E_B \) = encryption with A’s or B’s public key
\( K_{ses} = h(N_A, N_B) \)

Analysis:
- Session key secret and fresh; entity authentication ok
- **Session key not bound to A**

What is a protocol flaw?

- Researchers like to present spectacular attacks and flaws but the reality is often less black and white.

- Limitations on the applicability of the protocol:
  - Can the protocols withstand insider attacks?
  - Should multiple parallel executions be allowed with the same peer?
  - Is the protocol used for its original purpose or for something different?

- Requirements for implementations:
  - Encryption mode in old protocols is often assumed to protect integrity (MAC or non-malleable encryption)
  - Signed and MAC’d messages must include type tags and be parsed unambiguously

- New attacks and requirements arise over time:
  - Man-in-the-middle attacks
  - DoS protection, identity protection
  - What next? E.g. firewall traversal, mobility support, load balancing in server farms, distribution to the cloud, session migration, ...
Kerberos authentication
Kerberos

Shared-key protocol for user login authentication
- Uses passwords as shared keys
- Solves security and scalability problems in password-based authentication in large domains
- Based loosely on the Needham-Schroeder secret-key protocol

Kerberos v4 1988- at MIT
Kerberos v5 1993- [RFC 4120]
- Updated protocol and algorithms
- ASN.1 BER message encoding
- Implemented in Windows 2000 and later

Used in intranets: e.g. university Unix systems, corporate Windows domains
Kerberos architecture (1)

1. – 2. Authentication
3. – 4. Ticket for a specific service
4. – 5. Authentication to the service
Kerberos architecture (2)

1. KRB_AS_REQ
2. KRB_AS_REP
3. KRB_TGS_REQ
4. KRB_TGS_REP
5. KRB_AP_REQ
6. KRB_AP_REP

1.—2. Authentication with password → client gets TGT and K_{AT}
3.—4. Authentication with TGT and K_{AT} → client gets service ticket and K_{AB}
4.—5. Authentication with service ticket and K_{AB} → client gets service access

TGT, K_{AT}
Service ticket, K_{AB}
Kerberos ticket

- Same format for both TGT and service ticket
- Credentials = ticket + key
- ASN.1 encoding in Kerberos v5
- “Encryption” also protects integrity, actually encryption and a MAC

Flags:
- FORWARDABLE, FORWARDED, PROXIABLE, PROXY, MAY-POST-DATE, POSTDATED, INVALID, RENEWABLE, INITINIAL, PRE-AUTHENT, HW-AUTHENT
- INITIAL flag indicates TGT

<table>
<thead>
<tr>
<th>Message type, version</th>
</tr>
</thead>
<tbody>
<tr>
<td>REALM, SNAME</td>
</tr>
<tr>
<td>Server name and realm</td>
</tr>
<tr>
<td>FLAGS</td>
</tr>
<tr>
<td>KEY</td>
</tr>
<tr>
<td>CNAME, CREALM</td>
</tr>
<tr>
<td>Client name and realm</td>
</tr>
<tr>
<td>TRANSITED</td>
</tr>
<tr>
<td>transit realms</td>
</tr>
<tr>
<td>AUTH-TIME, END-TIME</td>
</tr>
<tr>
<td>CADDR</td>
</tr>
<tr>
<td>Client IP address (optional)</td>
</tr>
<tr>
<td>AUTORIZATION-DATA</td>
</tr>
<tr>
<td>App-specific access constraints</td>
</tr>
</tbody>
</table>

Encrypted with server’s master key
Protocol details

Initial login of user A:
1. A → AS: **Preauthentication**, A, TGS, N_{A1}, Addr_A
2. AS → A: A, TGT, E_{KA} (K_{A-TGS}, N_{A1}, TGS, Addr_A)

Ticket request:
3. A → TGS: TGT, Authenticator_{A-TGS}, B, N_{A2}, Addr_A
4. TGS → A: A, Ticket, E_{KA-TGS} (K_{AB}, N_{A2}, B, Addr_A)

Authentication to server B:
5. A → B: Ticket, Authenticator_{AB}
6. B → A: AP_REP

\(K_A\), \(K_{TGS}\), \(K_B\) = master keys of A, TGS and B
\(K_{A-TGS}\) = shared key for A and TGS
\(K_{AB}\) = shared key for A and B
\(TGT = B, E_{K_{TGS}} (INITIAL, K_{A-TGS}, A, T_{auth}, T_{expiry1}, Addr_A)\)
\(Ticket = B, E_{K_B} (K_{AB}, A, T_{auth}, T_{expiry2}, Addr_A)\)
**Preauthentication** = \(E_{KA} (^1T_A)\)
**Authenticator_{A-TGS}** = \(E_{KA-TGS} (^2T_A)\)
**Authenticator_{AB}** = \(E_{KAB} (^3T_A)\)
**AP_REP** = \(E_{KAB} (^4T_A)\)
Addr_A = A’s IP addresses

Notes:

1234) ASN.1 encoding adds type tags to all messages
Encryption mode also protects message integrity
Kerberos realms

Users and services registered to one KDC form a **realm**
- name@realm, e.g. A@X, alice@asia.sales.contoso.com

**Cross-realm trust:**
- Two KDCs X and Y share a key (krbtgt@Y is registered in KDC X and krbtgt@X in KDC Y)
- KDCs believe each other to be honest and competent to name users in their own realm

**Cross-realm authentication:**
- Client A@X requests from TGS at realm X a ticket for TGS at realm Y
- The ticket is encrypted for krbtgt@Y (i.e. TGS at realm Y)
- Client A@X requests from TGS at realm Y a ticket for server B@Y

**Access control at several steps:**
- Local policy at each KDC about when to honor tickets from other realms
- Local policy at B@Y about whether to allow access to users from other realms
- ACLs at B@Y determine whether the users is allowed to access the particular resources

**Possible to transit multiple realms → TRANSITED field in the ticket lists intermediate realms**
- Local policy at each server about which transit realms are allowed
Large organizations can have a **realm hierarchy**

- Hierarchy follows internet names
  - easy to find a path between realms
  - can filter cross-realm requests based on the names
- Can add **shortcut links** or create even a fully connected graph between KDCs
- E.g. Windows domain hierarchy

**Compare with X.509 certification hierarchy: similarities, differences?**
Password guessing attacks

Kerberos is vulnerable to password guessing:

- Sniffed KRB_AS_REQ or KRB_AS_REP can be used to test candidate passwords → **offline brute-force password guessing**
- In Kerberos v4, anyone could request a password-encrypted TGT from AS → easy to obtain material for password cracking
- **Preauthentication** in Kerberos v5 prevents active attacks to obtain material for password cracking → must sniff it

**Note: active vs. passive attacks**

- Misleading thinking: active attacks (e.g. MitM) are more difficult to implement than passive attacks (sniffing)
- **Reality:** Active attacks can often be initiated by the attacker while passive attacks require attacker to wait for something to sniff → vulnerability to such active attacks is serious
PKINIT

- Goal: take advantage of an existing PKI to bootstrap authentication in Kerberos
- Replaces the KRB_AS_REQ / KRB_AS_REP exchange with a public-key protocol
  - Public-key authentication and encryption to obtain TGT
  - Continue with standard Kerberos → transparent to TGS and application servers
- No password, so not vulnerable to password guessing
- Uses DSS signatures and ephemeral DH
- Windows 2000 and later, no standard [RFC 4556]
Delegation

- Server may need to perform tasks independently on the client’s behalf
  - Recursive RPC; agents operating when the user is no longer logged in; batch processing at night
- Alice can give her TGT or service ticket and key to David
- Controlling the use of delegated rights in applications:
  - Ticket may specify many allowed client IP addresses
  - Authorization-data field in ticket may contain app-specific restrictions
  - It is safer to delegate a service ticket than TGT
- Ticket flags related to delegation:
  - FORWARDABLE flag in TGT: the TGT can be used to obtain a new TGT with different IP addresses
  - PROXIABLE flag in TGT: the TGT can be used to obtain service tickets with a different IP address
- Kerberos delegation is identity delegation
  - When B has A’s ticket and key, B can act as A and nobody can tell the difference → difficult to audit access; similar to sharing passwords
Kerberos in Windows domains

Thanks to Dieter Gollmann
# Kerberos ticket in Windows

<table>
<thead>
<tr>
<th>Message type, version</th>
</tr>
</thead>
<tbody>
<tr>
<td>REALM, SNAME</td>
</tr>
<tr>
<td>Server name and realm</td>
</tr>
<tr>
<td>FLAGS</td>
</tr>
<tr>
<td>KEY</td>
</tr>
<tr>
<td>CNAME, CREALM</td>
</tr>
<tr>
<td>Client name and realm</td>
</tr>
<tr>
<td>TRANSITED</td>
</tr>
<tr>
<td>transit realms</td>
</tr>
<tr>
<td>AUTH-TIME, END-TIME</td>
</tr>
<tr>
<td>CADDR</td>
</tr>
<tr>
<td>Client IP address (optional)</td>
</tr>
<tr>
<td>AUTORIZATION-DATA</td>
</tr>
<tr>
<td>App-specific access constrains</td>
</tr>
</tbody>
</table>

- **Username, domain**
- **User and group SIDs**

Encrypted with server's master key
Puzzle of the day

(Hopefully most of you already know the answer to this puzzle.)

- **Diffie-Hellman protocol based on a commutative arithmetic operation:**
  - A chooses a random \( x \).
  - B chooses a random \( y \).
  1. \( A \to B: A, g^x \)
  2. \( B \to A: B, g^y \)
  - A calculates \( K_{ses} = (g^y)^x = g^{xy} \).
  - B calculates \( K_{ses} = (g^x)^y = g^{xy} \).

- **Security based on the computational Diffie-Hellman assumption:** given \( g, g^x \) and \( g^y \), it is infeasible to solve \( g^{xy} \)
  (Note: The calculations are done in some pre-agreed finite cyclic group where \( g \) is a generating element.)

- Looks simple and attractive, but what is wrong with this key exchange? Write down the attack
Related reading

- Kaufmann, Perlman, Speciner. Network security, 2nd ed.: chapter 14
- Ross Anderson. Security Engineering, 2nd ed.: chapter 3
- Dieter Gollmann. Computer Security, 2nd ed.: chapter 12.4
Exercises

- Find source code for a Kerberized client/server application (e.g. telnet) and see how it accesses Kerberos services.

- Why is Kerberos used on the intranets and TLS/SSL on the Internet? Could it be the other way?