Many-to-Many Authentication

How do users prove their identities when requesting services from machines on the network?

**Naïve solution**: every server knows every user’s password

- **Insecure**: break into one server \(\rightarrow\) compromise all users
- **Inefficient**: to change password, user must contact every server
Requirements

• Security
  – ... against attacks by passive eavesdroppers and actively malicious users

• User-friendliness
  – Transparent authentication process
  – Easy for legitimate user to change passwords

• Scalability
  – Large number of users and servers
Threats (an incomplete list)

• User impersonation
  – Malicious user with access to a workstation pretends to be another user from the same workstation

• Network address impersonation
  – Malicious user changes network address of his workstation to impersonate another workstation

• Eavesdropping, tampering, replay
  – Malicious user eavesdrops, tampers, or replays other users’ conversations to gain unauthorized access
Solution: Trusted Third Party

- Trusted **authentication service** on the network
  - Knows all passwords, can grant access to any server
  - Convenient (but also the single point of failure!)
  - Requires high level of physical security
What Should a Ticket Look Like?

- User should not be able to access server without first proving his identity to authentication service.
- Ticket proves that user has authenticated.
  - Authentication service encrypts some information with a key known to the server (but not the user!)
    - The only thing the user can do is pass the ticket to the server.
    - Hash functions would’ve worked well, but this is 1980s design.
  - Server decrypts the ticket and verifies information.

**Ticket** gives the holder access to a network service.
What Should a Ticket Include?

- User name
- Server name
- Address of user’s workstation
  - Otherwise, a user on another workstation can steal the ticket and use it to gain access to the server
- Ticket lifetime
- A few other things (session key, etc.)
Naïve Authentication

- **Insecure:** passwords are sent in plaintext
  - Eavesdropper can steal the password and later impersonate the user to the authentication server
- **Inconvenient:** need to send the password each time to obtain the ticket for any network service
  - Separate authentication for email, printing, etc.
Two-Step Authentication

- Prove identity **once** to obtain a special **TGS ticket**
- Use TGS to get tickets for any network service

Joe the User

- User: `USER=Joe; service=TGS`
- Encrypted TGS ticket
- TGS ticket
- Encrypted service ticket

Key distribution center (KDC)

Ticket granting service (TGS)

File server, printer, other network services
Threats

• Ticket hijacking
  – Malicious user may steal the service ticket of another user on the same workstation and try to use it
    • Network address verification does not help
      – Servers must verify that the user who is presenting the ticket is the same user to whom the ticket was issued

• No server authentication
  – Attacker may misconfigure the network so that he receives messages addressed to a legitimate server
    • Capture private information from users and/or deny service
      – Servers must prove their identity to users
Symmetric Keys in Kerberos

- $K_c$ is long-term key of client C
  - Derived from the user’s password
  - Known to the client and the key distribution center (KDC)
- $K_{TGS}$ is long-term key of TGS
  - Known to KDC and the ticket granting service (TGS)
- $K_v$ is long-term key of network service V
  - Known to V and TGS; each service V has its own long-term key
- $K_{c,TGS}$ is short-term session key between C and TGS
  - Created by KDC, known to C and TGS
- $K_{c,V}$ is short-term session key between C and V
  - Created by TGS, known to C and V
“Single Logon” Authentication

**User**
- **password**

**kinit program (client)**
- **$K_c$**
  - Decrypts with $K_c$ and obtains $K_{c,TGS}$ and $ticket_{TGS}$

**Key Distribution Center (KDC)**
- **ID$_c$, ID$_{TGS}$, time$_c$**
- **Encrypt$_{K_c} (K_{c,TGS}, ID_{TGS}, time_{KDC}, ticket_{TGS})$**
  - Fresh key to be used between client and TGS
- **Encrypt$_{K_{TGS}} (K_{c,TGS}, ID_c, Addr_c, ID_{TGS}, time_{KDC}, lifetime)$**
  - Client will use this unforgeable ticket to get other tickets without re-authenticating

**Implicit authentication:** only someone who knows $K_c$ can decrypt

- Client only needs to obtain TGS ticket *once* (say, every morning)
- Ticket is encrypted; client cannot forge it or tamper with it
Obtaining a Service Ticket

- Client uses TGS ticket to obtain a service ticket and a **short-term session key** for each network service (printer, email, etc.).
Obtaining Service

- For each service request, client uses the short-term key for that service and the ticket he received from TGS.
Kerberos in Large Networks

• One KDC isn’t enough for large networks (why?)
• Network is divided into realms
  – KDCs in different realms have different key databases
• To access a service in another realm, users must...
  – Get ticket for home-realm TGS from home-realm KDC
  – Get ticket for remote-realm TGS from home-realm TGS
    • As if remote-realm TGS were just another network service
  – Get ticket for remote service from that realm’s TGS
  – Use remote-realm ticket to access service
  – \( N(N-1)/2 \) key exchanges for full \( N \)-realm interoperation
Summary of Kerberos

1. as_request: "Hi, I'm John. Can I have a ticket for getting tickets?"

2. as_reply: "Here's a ticket-granting ticket, encrypted with John's secret key".

3. tgs_request: "Here is my TGT, could I have a ticket for Service A?"

4. tgs_reply: "Here's a ticket for Service A."

5. ap_request: "Here is my ticket; let me use your service."

6. ap_reply: "Welcome John! By the way, here's the proof that I'm Service A."
Important Ideas in Kerberos

• Short-term session keys
  – Long-term secrets used only to derive short-term keys
  – Separate session key for each user-server pair
    • Re-used by multiple sessions between same user and server

• Proofs of identity based on authenticators
  – Client encrypts his identity, addr, time with session key;
    knowledge of key proves client has authenticated to KDC
    • Also prevents replays (if clocks are globally synchronized)
  – Server learns this key separately (via encrypted ticket that
    client can’t decrypt), verifies client’s authenticator

• Symmetric cryptography only
Kerberos Version 5

• Better user-server authentication
  – Separate subkey for each user-server session instead of re-using the session key contained in the ticket
  – Authentication via subkeys, not timestamp increments
• Authentication forwarding (delegation)
  – Servers can access other servers on user’s behalf, e.g., can tell printer to fetch email
• Realm hierarchies for inter-realm authentication
• Explicit integrity checking + standard CBC mode
• Multiple encryption schemes, not just DES
Practical Uses of Kerberos

• Microsoft Windows
• Email, FTP, network file systems, many other applications have been kerberized
  – Use of Kerberos is transparent for the end user
  – Transparency is important for usability!
• Local authentication
  – login and su in OpenBSD
• Authentication for network protocols
  – rlogin, rsh
• Secure windowing systems
Charge

• Kaufman Chapters 13 and 14
• “Designing an Authentication System: A Dialogue in Four Scenes”
  – A high-level survey of network threats and design principles behind Kerberos