Kerberos

Haipeng Dai

haipengdai@nju.edu.cn
313 CS Building
Department of Computer Science and Technology
Nanjing University
The Problem

- Problem:
  - Multiple users
  - Multiple client computers
  - Multiple server computers

we need
1. Authentication
2. Authorization
3. Confidentiality
4. Freshness
5. Auditing

- How should we do authentication here?
  - Users need to prove their identities when requesting services at servers from client machines.
Kerberos

- Kerberos is an **authentication** and **authorization** protocol
- Kerberos uses a **trusted third party** authentication service that enables clients and servers to establish authenticated and secure communication.
- Kerberos provides **single sign-on** capability using a centralized repository of accounts.
- Relies entirely on **symmetric cryptography**
- Kerberos provides an **audit trail of usage**.
  - How? You should be able to answer after this lecture.
- Developed at MIT: two versions, Version 4 and Version 5 (specified as RFC1510)
- [http://web.mit.edu/kerberos/www](http://web.mit.edu/kerberos/www)
- Used in many systems, e.g., Windows 2000 and later as default authentication protocol
- In Greek mythology, Kerberos means a many headed dog, commonly three, perhaps with a serpent's tail, the guardian of the entrance of Hades."
  - the modern Kerberos was intended to have three components to guard a network's gate: **authentication**, **accounting**, and **audit**.
Requirements

- **Security**
  - A network eavesdropper should not be able to obtain the necessary information to impersonate a user.

- **Transparency**
  - Users shouldn’t notice authentication taking place beyond the requirement to enter a password.

- **Scalability**
  - The system should be capable of supporting large numbers of clients and servers.

- **Reliability**
  - For all services that rely on Kerberos for access control, lack of availability of the Kerberos service means lack of availability of the supported services.
Threat Model

- **User impersonation**
  - A user may gain access to a particular workstation and pretend to be another user operating from that workstation.

- **Network address impersonation**
  - A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.

- **Eavesdropping, tampering and replay**
  - A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations.
Straightforward Solution 1

- Solution 1: Each client handles authentication & authorization by itself using its database.

- Drawbacks:
  - All user databases need to be synchronized across all client PCs. Management tasks like adding or deleting a user need to be performed on every client PC. Obviously this solution is not scalable with the number of client PCs.
Straightforward Solution 2

- Solution 2: Each server handles authentication & authorization by itself using its database.

- Drawbacks:
  - All user databases need to be synchronized across all servers. Management tasks like adding or deleting a user need to be performed on every server. This solution is not scalable with the number of servers. It is hard to manage user accounts and access rights.
Kerberos

- **Key management:**
  - Every server does have a shared key with AS
  - Every user has a password with AS
  - Every client does NOT have a shared key with AS
    - Clients mean the client PCs. Users mean human users.
    - Why?
      - We want the clients to be “dummies”. This not only makes the management of clients easier but also makes the system secure against client compromises.
Kerberos: Alex Version 1

- If every client has a shared key with AS, then we can use the symmetric key based authentication protocol using trusted third party.
  
  - Note: Kerberos versions 1-3 were not published. This is “Alex” version of Kerberos Version 1 for teaching purposes.
Kerberos: Alex Version 2

- Why does the ticket include Address\(_C\)?
  - Otherwise another malicious client can steal a ticket and replay it.
  - Yes, an attacker can still replay it by changing the address of a compromised client, but he has to wait client C to power off.

- Why encrypted with K\(_V\)?
  - Prevent ticket from being forgeable.

- Defect of this protocol:
  - Password of U is sent in plaintext.
  - How to encrypt user U’s password?
    - Since both Client C and AS knows U’s password, they can generate a key from MD(PASS\(_U\)).
Kerberos: Alex Version 3 (1/2)

- **Step 1: Authentication**
  - Once per logon session. AS needs to search in user database for \( \text{PASS}_U \).

- **Step 2: Authorization**
  - Once per type of service.
    - TGS needs to search in authorization database or access control rules.
  - Once per service session.
Kerberos: Version 3 (2/2)

- Why not combine these two steps?
  - If we do, then for each server that U wants to access, AS needs to search in its user database and authorization database.

- Why use timestamp?
  - Otherwise, an attacker can steal the ticket, wait for C to power off, and reconfigure his client using C’s address, and then replay the ticket.
Defects of Version 3

- Major Defect 1: Only one-way authentication. Client C cannot authenticate AS, TGS, or V.
  - How to achieve mutual authentication? Think: What can be used as a shared secret? To achieve mutual authentication, there must be some secret shared.
    - U’s password is the shared secret between C and AS.
  - Then what about C and TGS? What can be used as a shared secret?
    - We can let AS to create one for C and TGS.
  - Then what about C and V? What can be used as a shared secret?
    - We can let TGS to create one for C and V.
Defects of Version 3

- Major Defect 2: TGS cannot know whether 
  \( \{U|\text{Address}_C|\text{TGS}|TS_1|\text{Lifetime}_1\}_{K_{tg}} \) is a replayed one.
  - If lifetime is too short: ASK AS or TGS too often.
  - Similarly, if lifetime is too long, V cannot know whether 
    \( \{U|\text{Address}_C|V|TS_2|\text{Lifetime}_2\}_{K_V} \) is a replayed one.
  - Difficult to choose the right lifetime

- How to solve this problem:
  - In addition to showing me your ticket, prove to me your identity each time you talk to me.
    - C generates an authenticator each time it talks to TGS or V.
Kerberos: Version 4

- **Authentication**
  - AS needs to make sure C is not much out of synchronization

  \[ \{K_{C, tgs} | TGS | TS_2 | Lifetime_2 \} K_{tgs} \]

  for C to authenticate AS

- **Authorization**
  - Client C

  \[ \{K_{C, tgs} | U | Address_C | TGS | TS_2 | Lifetime_2 \} K_{tgs} \]

  for C to authenticate TGS

- **Authentication & Authorization**
  - I believe that this is a typo in the original paper

  \[ \{K_{C, v} | U | Address_C | V | TS_4 | Lifetime_4 \} K_{V} \]

  Messages afterwards may or may not be encrypted by the session key \( K_{C,V} \), depending on the application.
Kerberos: Version 4

- Why do we need authenticator?
  - To prevent message replay. Although replaying the TGS ticket does not allow attackers to get a fresh server ticket because attackers does not know $K_{C\text{-}tg}$, it may cause wrong audit record of user $U$ uses service $V$.
  - Authenticator is generated very recently.
  - Even if an attacker steals a ticket, because he does not know the shared key between $C$ and TGS and the shared key between $C$ and $V$, he cannot generate a fresh authenticator.

- Loose synchronization:
  - If TGS current time – $TS_3 >$ maximum clock difference between TGS and $C$, then the authentication fails.
  - If $V$’s current time – $TS_5 >$ maximum clock difference between $V$ and $C$, then the authorization fails.

- The textbook and the original Kerberos paper did not include Lifetime4 in the message from TGS to $C$ outside the server ticket in the protocol specification. But I think it should be included because otherwise $C$ does not know when the server ticket will expire. I believe that this is a typo in the original paper because in the paper the authors said that the format of message from TGS to $C$ is identical to the format of the message from AS to $C$. 
A Kerberos realm is a set of managed nodes that share the same Kerberos database.
Problems of Version 4

- **Nonscalable for interrealm authentication**
  - n realms need $n^2$ Kerberos-to-Kerberos relationships
  - Solution: use a hierarchy of realms (i.e., make a tree).

- **Double encryption: tickets are encrypted twice.**
  - Solution: move the ticket out of the encryption.

- **Session key**
  - Used by both authenticator and subsequent message encryption cross multiple sessions between C and V. Thus, an attacker may replay a message encrypted by $K_{C,V}$ in a previous session as a message encrypted by $K_{C,V}$ in a later session.
  - Solution: using randomly started sequence numbers.
Problems of Version 4

- Engineering problems
  - Encryption depends on DES → Encryption keys are tagged with a type and a length
  - Rely on IP address → Network addresses are tagged with a type and a length
  - Ticket lifetime is 8-bit length. Unit is 5 min and maximum is 21 hours.
    → Ticket includes a start time and ended time
  - Message byte ordering: indicates least significant bytes in lowest/highest address by a tag. It is not convenient.
    → All messages are defined using Abstract Syntax Notation One (ASN.1) and Basic Encoding Rules (BER)
  - PCBC encryption: version 4 uses a non standard mode of DES, i.e., Propagating Cipher Block Chaining (PCBC). This mode is insecure.
Kerberos: Version 5 (1/2)

- Authentication

  Options|U|Realm_U|TGS|TimeSetting|n_1

  Realm_U|U|\{Flags|K_{C,tgs}|Realm_U|U|Address_C|TimeSetting\}_{K_{tgs}} Ticket_{tgs}

  |\{K_{C,tgs}|TimeSetting|n_1|Realm_{tgs}\|TGS\}_{MD(PASS_U)}

- Authorization

  Options|V|TimeSetting|n_2|\{Flags|K_{C,v}|Realm_U|U|Address_C|TimeSetting\}_{K_{tgs}}

  Realm_U|U|\{Flags|K_{C,v}|Realm_U|U|Address_C|TimeSetting\}_{K_{v}} Ticket V

  |\{K_{C,v}|TimeSetting|n_2|Realm_U\|V\}_{K_{C,tgs}}

- Authentication & Authorization

  Options|\{Flags|K_{C,v}|Realm_U|U|Address_C|TimeSetting\}_{K_{v}}

  |\{U|Realm_U|TS_2\}_{K_{C,v}}

  \{TS_2|Subkey|Seq#\}_{K_{C,v}}
Kerberos: Version 5 (2/2)

A.com Realm

Client C

User U

Request ticket for local TGS

Ticket for local TGS

Request ticket for com TGS

Ticket for com TGS

Kerberos Server

Remote service

Request ticket for B TGS Ticket

B TGS Ticket

Server V

Kerberos Server

com Realm

Request ticket for remote server V

Kerberos Server

B.com Realm
Password-to-Key transformation (1/2)

- Convert password to bit stream

- Convert bit stream to input key
Password-to-Key transformation (2/2)

- Generate final key using DES in CBC mode
MS Windows

- Domain login
- Kerberos Ticket (Windows Kerbtray.exe application)
- Notice realm - FERMI.WIN.FNAL.GOV
MS Windows Managing Credentials

- MIT Kerberos for Windows (KfW)  
  http://web.mit.edu/kerberos/

- Notice realm - FNAL.GOV
MS Windows Managing Credentials

- WRQ Kerberos Manager
MS Windows Managing Credentials

- OpenAFS Token
UNIX, Linux, Mac OS X

- **Kerberos tools:**
  - kinit
  - klist
  - kdestroy
  - k5push

- **Clients:**
  - telnet, ssh, ftp
  - rlogin, rsh, rcp

```
Last login: Wed Mar 2 08:23:53 on console
Welcome to Darwin!
[rreitz@brevix rreitz]$ kinit
Please enter the password for rreitz@FNAL.GOV:
[rreitz@brevix rreitz]$ klist -f
Kerberos 5 ticket cache: 'API:Initial default ccache'
Default Principal: rreitz@FNAL.GOV
Valid Starting       Expires            Service Principal
03/02/05 13:53:15    03/03/05 15:53:14   krbtgt/FNAL.GOV@FNAL.GOV
                   renew until 03/09/05 13:53:14, FPRIA

[rreitz@brevix rreitz]$ ssh fenris
                                 NOTICE TO USERS
[rreitz@fenris rreitz]$ klist -f
Ticket cache: /tmp/krb5cc_3819_5Zmg4N
Default principal: rreitz@FNAL.GOV
Valid starting       Expires            Service principal
03/02/05 14:54:21    03/03/05 15:53:14   krbtgt/FNAL.GOV@FNAL.GOV
                   renew until 03/09/05 13:53:14, Flags: FfPRA
03/02/05 14:54:21    03/03/05 15:53:14   afs@FNAL.GOV
                   renew until 03/09/05 13:53:14, Flags: FfPRA
[rreitz@fenris rreitz]$
```