

The Kerberos Authentication System

Brad Karp
UCL Computer Science



CS GZ03 / M030
23rd November 2011

Why Study Kerberos?

- One of **most widely used authentication systems**, implemented in many, many UNIXes for a variety of services
- Simple example of use of cryptography to solve **practical authentication problems**
- Imperfect; **weaknesses instructive**

Kerberos: Goals

- Authentication of **diverse entities, for diverse services**:
 - Users, client machines, server machines
 - File systems, remote login, file transfer, printing, &c.
- Authentication in an **“open environment”**
 - Users may be **superuser on their own workstations** (so may adopt **any user ID** if credentials over network unauthenticated); **hardware not centrally controlled**
 - Same user population may use **many machines and services** (e.g., labs of public-access machines on a campus)
- Drop-in replacement of passwords for pre-existing protocols
 - **Convenient; strength of security?**

Kerberos Model: Central Authority

- Within a site (e.g., MIT), a central database server stores **names and secret keys for all principals**
 - Keys are for 56-bit DES symmetric-key cipher
 - Now **brute-forceable**; more reasonable at time of Kerberos' first use (1988)
- All users and machines are principals, named with **human-readable names**
- All principals **trust central database server**

Kerberos Principal Names

- Users: e.g., bkarp
 - Can have **instances**; sub-names of a principal, e.g., bkarp.mail, bkarp.root
- Machines: e.g., boffin, arkell, sonic
- Services: e.g., rlogin.sonic (instance of the rlogin service running on sonic)
- Site name: **realm**; all machines in one administrative domain share one central Kerberos database, in same realm
- name.instance@realm, e.g., bkarp@UCL.AC.UK

Kerberos Protocol

- Goal: **mutually authenticated communication**
 - Two principals wish to communicate
 - Principals know each other by name in central database
 - Kerberos establishes shared secret between the two
 - Can use shared secret to encrypt or MAC subsequent communication
 - [Few “Kerberized” services encrypt, and **none MAC!**]
- Approach: leverage keys shared with central database
 - Central database **trusted by/has keys for all principals**

Kerberos Credentials

- Client can either be user or machine, depending on context
- To talk to server s , client c needs **shared secret key** and **ticket**:
 - **Session key**: $K_{s,c}$ (randomly generated by central database)
 - **Ticket**:
 $T = \{s, c, \text{addr}_c, \text{timestamp}, \text{lifetime}, K_{s,c}\}_{K_s}$
(where K_s is key s shares with database)
 - Only server s can decrypt ticket

Kerberos Credentials (2)

- Given ticket, client creates authenticator:
 - Authenticator:
 $A = \{c, \text{addr}_c, \text{timestamp}\}_{K_{s,c}}$
 - Client must know $K_{s,c}$ to create authenticator
 - Authenticator can **only be used once**
- Client presents both ticket T and authenticator A to server when requesting an operation
 - T convinces server that $K_{s,c}$ was given to c
 - A intended to **prevent replay of requests**
- “Kerberized” protocols use authenticator in place of password

Getting the User's First Ticket

- User logs in at console with username and password (username is Kerberos name)
- Kerberized login program retrieves initial ticket for user:
 - Client machine sends to Kerberos database:
 c, tgs
(tgs is principal name for [ticket-granting service](#))
 - Server responds with:
 $\{K_{c,tgs}, \{T_{c,tgs}\}_{K_{tgs}}\}_{K_c}$
 - where
 $T_{c,tgs} = tgs, c, addr_c, timestamp, lifetime, K_{c,tgs}$
 - Client decrypts server's response with
 $K_c = H(\text{password})$

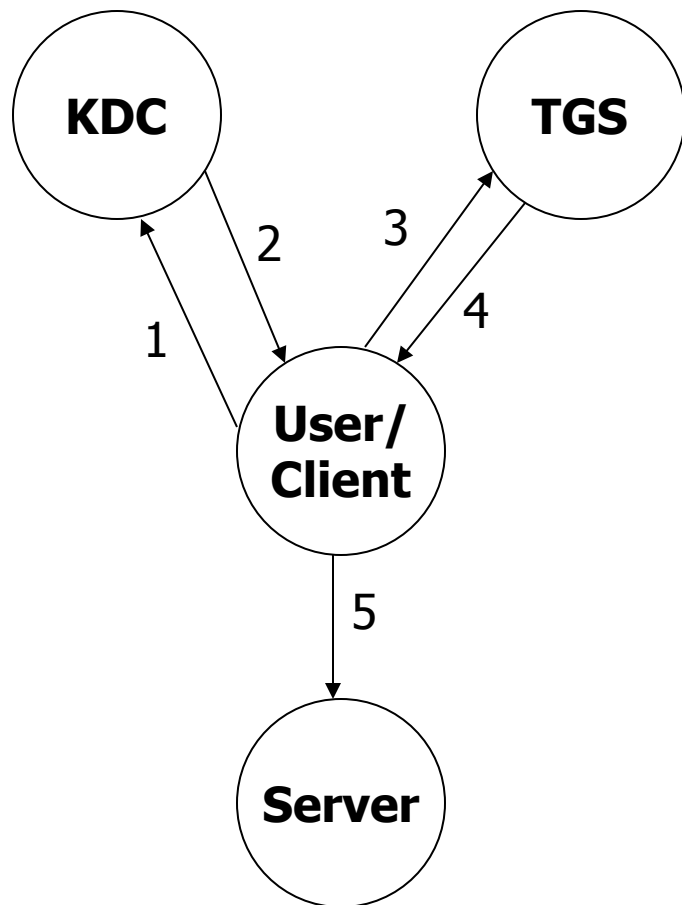
Requesting a Service

- Client c (e.g., user bkarp) wishes to use a service on s , already holds $K_{c,tgs}$
- Client requests ticket from tgs as follows:
 - Client sends to tgs:
 $s, \{T_{c,tgs}\}_{K_{tgs}}, \{A_c\}_{K_{c,tgs}}$
 - tgs replies to client with ticket for service on that server:
 $\{\{T_{c,s}\}_{K_s}, K_{c,s}\}_{K_{c,tgs}}$
 - where $K_{c,s}$ is a **new, randomly generated session key** for use between c and s

Using a Service

- Once client holds ticket for service, uses it with authenticator to request operation from server:
 - Client sends to s:
service name, $\{T_{c,s}\}_{K_S}, \{A_c\}_{K_{C,S}}$
 - Server validates $T_{c,s}$ and A_c , and executes operation if they are valid
- Server uses timestamps and expiration times to **invalidate stale, “future”, replayed requests**

Kerberos: Summary of Message Flow



1. Request for TGS ticket:
 c, tgs
2. Ticket for TGS:
 $\{K_{c,tgs}, \{T_{c,tgs}\}_{K_{tgs}}\}_{K_c}$
3. Request for Server ticket:
 $s, \{T_{c,tgs}\}_{K_{tgs}}, \{A_c\}_{K_{c,tgs}}$
4. Ticket for Server:
 $\{\{T_{c,s}\}_{K_s}, K_{c,s}\}_{K_{c,tgs}}$
5. Request for Service:
 $service\ name, \{T_{c,s}\}_{K_s}, \{A_c\}_{K_{c,s}}$

Ticket Lifetime

- How should we choose ticket lifetimes?
- Convenience: longer ticket-granting ticket lifetime → user must type password less often
- Performance: longer service ticket lifetime → client must request new service ticket less often
- Risk: longer ticket lifetime lengthens period when ticket can be stolen, abused
- MIT Athena implementation destroys ticket-granting ticket when user logs out

Kerberos Security Weaknesses

- Vulnerability to **replay attacks**
- Reliance on **synchronized clocks across nodes**
- Storage of tickets on workstations
- No way to change compromised password securely
- Key database focal point for attack
- Hard to upgrade key database (relied on by all nodes in system)

Kerberos User Inconveniences

- Large (e.g., university-wide) administrative realms:
 - University-wide admins **often on critical path**
 - Departments **can't add users or set up new servers**
 - **Can't develop new services** without central admins
 - **Can't upgrade software/protocols** without central admins
 - Central admins have **monopoly servers/services** (can't set up your own without a principal)
- Rigid; what if user from realm A wants to authenticate himself to host at realm B?
- Ticket expirations
 - Must renew tickets every 12-23 hours
 - **How to create long-running background jobs?**