Kerberos Introduction

Jim Binkley- jrb@cs.pdx.edu

outline

- intro to Kerberos (bark, bark)
- protocols
  - Needham Schroeder
  - K4
  - K5
- miscellaneous issues
- conclusion

Kerberos history

- Kerberos came from MIT
- part of project Athena, in 1980s
  - which also developed the X window system
- Kerberos 4 released in 1989
  - used DES, therefore export-control prevented export of US release
  - Australian programmer took un-DES'ed form and produced his own DES, called system ebones
- K4 can be considered dead, but maybe not ...

k-istory, cont

- why Kerberos, the name?
  - because Cerberus was a vicious beast that guarded the gates of hell
  - 3 dog heads, and a dragon tail
  - one had to “authenticate” to pass into hell
    - or escape ...
  - it should be noted however that the hero Hercules kidnapped Cerberus ...

is this an early DOS attack?

k-istory, cont

- Kerberos 5 intended to fix bugs, make improvements
  - likely what is used today
  - RFC 1510 plus supplements document it
    - » K4 only documented in code
  - protocol done in ASN.1
  - extensible encryption types
  - pre-authentication feature

k-istory, cont.

- MIT reference implementation for K-5
- Heimdal - open source version
- Windows 2000 and above from MS
  - public key extensions
- Apple also uses it
- IETF has been attempting to formalize it

more info

- ORA - Kerberos book. Jason Garman
  - August 2003
  - practical setup/debugging info
- Network Security, KRS
  - 2 chapters
- MIT Dialogue in Four Scenes: *web.mit.edu/Kerberos/www/dialogue.html*

more info 2:

- MIT home page: web.mit.edu/Kerberos/www
- Heimdal home page: www.pdc.kth.se/heimdal
- paper: Neuman/Ts'o. *Kerberos: An Authentication Service for Computer Networks*, IEEE Communications, Sept. 1994
- paper: Bellovin/Merritt. *Limitations of the Kerberos Authentication System*, USENIX, 1991.

Basic concepts

- Kerberos basically authenticates clients to servers
- passwords never sent in the clear
  - we send “tickets” instead
- a ticket is an encrypted session-key with a timeout
- a “directory” may be used in an implementation to hold keys
  - e.g., MS has an LDAP directory structure

terminology

- principal - a kerberos user
  - may be service
  - may be person
- a principal is a name
  - K4 form:
  - user[.instance]@REALM
  - service.hostname@REALM

names, cont.

- because K4 did not allow two hosts with the same name in the same realm
- K5 principal like so:
  - username[/instance]@REALM
  - service/FQDN@REALM
- e.g.,,
  - host/foo.com@REALM
  - host/bar.com@REALM

a REALM

- a realm is the domain of a KDC
  - typically an enterprise or one admin domain
- realm name usually same as DNS
  - BUT UPPERCASE
  - joebob/admin@MYFOO.BAR.COM
- name doesn't have to be DNS though

Kerberos services

- passwords are not transmitted in the clear
  - and in fact, session-keys are sent
- single-sign-on
  - user logs in once, and can talk to multiple services without having to reverify with a password (possibly a different password)
- mutual authentication
  - alice/bob both authenticate to each other

the man behind the curtain

- we must have a KDC
  - better a *distributed* KDC
  - KDC had better be a very secure host
  - not on Inet ... minimal services, etc.
  - super Bastion Host ...
- we must issue passwords and both Alice and the KDC must know them

KDC has 3 parts

- database of principals and keys
  - MS uses LDAP
  - Heimdal puts in specialized db
- ticket-granting-server - takes care of ticket-granting for Alice/Bob (user/server) exchange
- authentication-server - implements single sign-on function
  - issues TGT (ticket granting ticket) that Alice's software can use to get individual tickets to talk to other servers

cont.

- ticket granting service has 2 inputs:
  - 1. the ticket granting ticket (TGT)
  - 2. principal name for desired service (bob)
- TGS verifies that TGT is valid
  - by decode with KDC symmetric key

a ticket is:

- user's principal - who wants the service
- service's principal - who does the service
- when started, and when becomes invalid
- list of IP addresses involved
- the shared secret key encrypted with a principal's key
- ticket's usually last hours or a day

Fundamental protocol

- Needham Schroeder protocol, Xerox, 1978
- Assume Alice, Bob, and KDC
  - key distribution center
- note: Bob may be a service
  - a printer, file system, telnet server, etc.
- Alice, Bob, and KDC all have symmetric secret keys
  - or passwords that can be turned into symmetric keys
- KDC has keys stored on it

algorithm underpinnings

- 1. a-priori shared secret between KDC and Alice/KDC and Bob
  - 2 master keys
- 2. Alice gets from KDC two session keys
  - 1. one encrypted for Alice with Alice's master
  - 2. one encrypted for Bob with Bob's master
  - 3. this is a new Alice/Bob session key
- 3. Alice send's Bob Bob's key, and Bob decrypts with Bob's master key

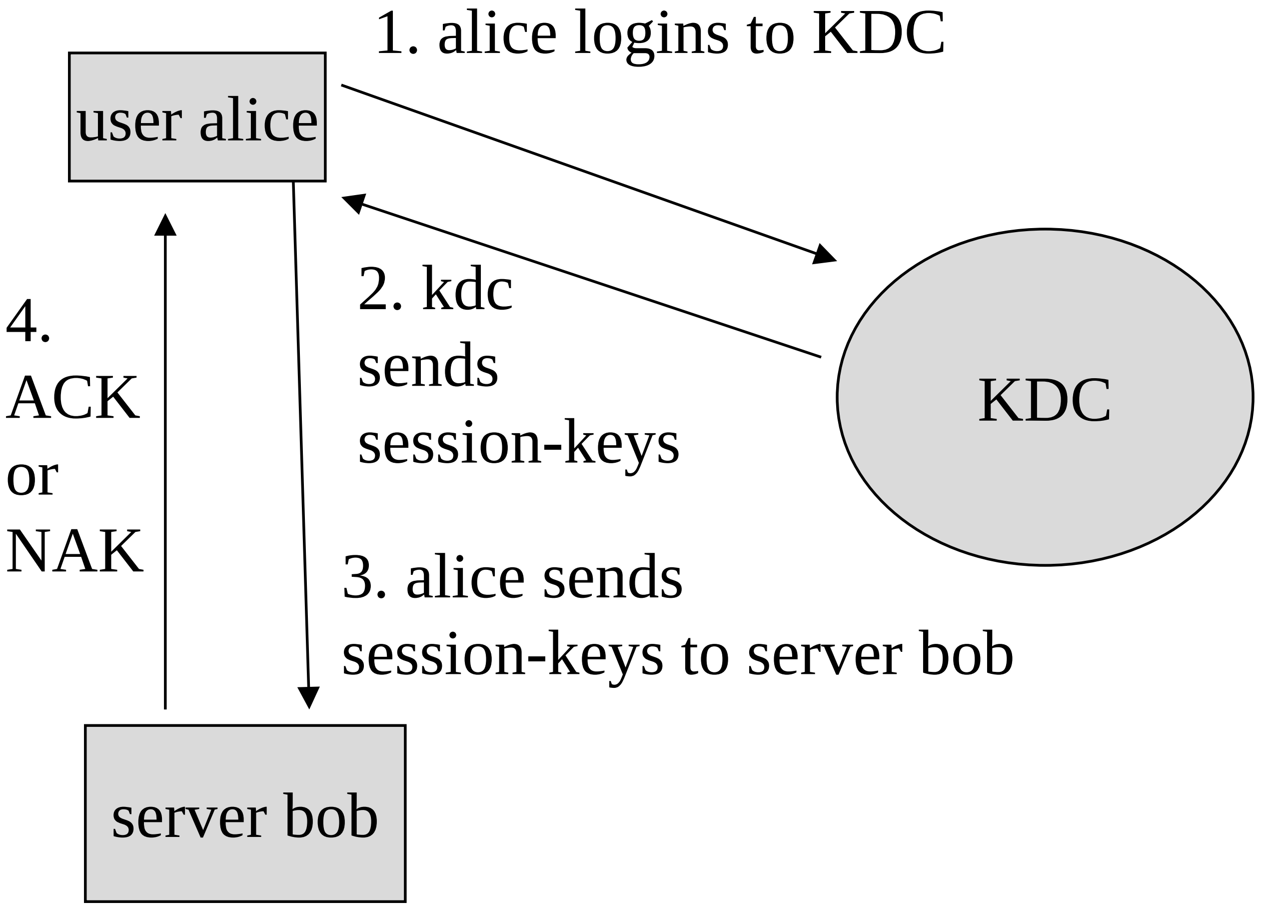
N/S cont.

- M1 to KDC : A wants to talk to B, N1
  - A can encrypt with her key
  - Nonce is included here to make sure KDC reply is fresh
- M2, KDC to A: Kalice{N1, Bob id, Kab, Bob ticket}
  - Kab is a session key, Bob ticket is the session key encrypted with Bob’s secret key
  - all encrypted with Alice’s secret key
  - Alice can’t make anything with Bob’s ticket as she doesn’t have Bob’s key

N/S cont.

- m3: Alice to Bob: ticket, challenge to Bob
  - challenge, has N2 encrypted with Kab.
  - Bob opens the ticket to get Kab, and can decode N2
  - ticket has Alice's name it in for mutual authentication
- m4: Bob to Alice: Kab{N2-1, N3}
- m5: Alice to Bob: Kab{N3-1}
- last two steps done for mutual authentication, and proof that they both know the secret key.
  - manipulate the nonce by subtracting one.

KDC picture


1. alice logins to KDC
user alice
2. kdc
sends
session-keys
KDC
4.
ACK
or
NAK
3. alice sends
session-keys to server bob
server bob

a number of holes exist

- passwords are imperfect ...
  – Alice may have a bad password
  – or may lose it
  – or may give it away
  – or the protocol itself as implemented might be subject to brute-force password cracking
  – e.g., what if a ticket is simply stored in a file and an attacker gets the file?

one improvement

- Alice 1st talks to Bob
- Bob sends Alice Kbob{Nonce of Bob}
- Alice sends this nonce to the KDC
- which puts it in Bob’s ticket
- this prevents Bad Bertha from using Alice’s old key, once Alice has changed her key
- Bob knows that the key Alice used with the KDC is post its own nonce being sent.

Kerberos 4 basic protocol

- two major changes
- 1. we assume shared time, which gets rid of the challenge-response protocol
  - e.g., we use NTP
- 2. in order to implement single-sign-on, we implement a ticket-granting server
  - authentication service (Alice to Bob)
  - ticket granting service (Alice to KDC)

K4 protocol

- part 1: authentication server
  - password from user turned into
  - ticket granting ticket
- part 2: ticket-granting server
  - TGT and principal info turned into
  - session key for Alice/Bob

authentication server function

- client sends AS_REQ: (client principal, client timestamp, krbtgt (ticket granting server principal name), requested lifetime)
  - sent in plaintext
  - probably at start of day
  - probably last 8-10 hours
  - krbtgt.hostname@REALM is TGS principal
- server must verify that time is within a certain limit, say 5 minutes

AS generates session key

- session key shared between Alice and TGS
  - one copy for client
  - one for TGS
- sends AS_REP message to client:
  (user copy of session key, krbtgt principal, ticket lifetime, TGS ticket)
  - all of this message encrypted with client secret key
  - TGS ticket encrypted with TGS secret key

AS reply continued.

- TGS “key box” (ticket) contains:
  - TGS copy of session key
  - client principal
  - ticket lifetime
  - KDC timestamp
  - client ip address
- this is cached at client
- client gets user password to decode ...
- thus we get single-sign-on

so client now has

- 1. a session key
- 2. a ticket-granting ticket
  - which it caches in a file or in memory
  - memory is probably a better idea, why?

part 2: ticket-granting server

- client sends TGS request: (service principal name, TG ticket, authenticator, timestamp)to Ticket Granting Server
- authenticator (encrypted with TGS session-key) : (timestamp, client principal)
  – client has knowledge of shared key
  – proves uniqueness of request
- KDC formulates reply

part 2: TGS reply

- TGS reply (encrypted with TGS session key): (user copy of new session key, service principal name, ticket lifetime, service ticket)
- service ticket (encrypted with service key): (service copy of new session key, client principal, ticket lifetime, KDC timestamp, client ip address)

client sends ticket to server

- this is not part of the K protocol
  - or this is app dependent
  - K system provides library code to help
- we might mount a windows file-share
  - or talk to a telnet daemon at this point

K4: some details:

- K4 requires us to take password string
  - e.g., create a 56-bit DES key
  - call this string2key()
  - similar to UNIX password function
- encryption is possible if app wants
  - provided in library
  - kerberos provides this format:
    (version, message type, length, cybercrud)
- in K4, this is DES in PCBC mode using session key

K4: some details

- integrity checking is possible
- KRS states that algorithm was an MIT variation on Jueneman MAC
  - kerberos calls MAC's "checksums"
  - not good practice, why?
  - K5 uses more commonly accepted algorithms

K5 overview

- ASN.1 (ouch, ouch, ouch, ouch)
  - means we can neglect protocol details
  - except when they bite us ...
- neglecting that all the protocol bits have changed, it can be viewed as similar
- but more extensible
  - K4 assumed DES! ... need more variation than that

K5 overview

- credential forwarding is one feature
  - user gets to serverA with telnet
  - now wants to ftp to serverB ...
  - with K4 can't do that
  - in K5, ticket-granting-ticket is sent to remove server upon login

ASN.1

- allows variable length forwarding in a
- TAG, LENGTH, VALUE format
- can view both as protocol and data definition language
- has basic types
- and constructed types made from basic types
- used in SNMP, certificate formats, LDAP, H323
- KRS points out IP address takes 15 bytes to encode!

K5 overview continued:

- K4 assumed DES
- K5 allows other choices, including entirely new choices (in case the previous one springs a leak)
  - as any good crypto protocol should
  - keys are tagged with type and length
  - rsa-md5-des is required (des is not a good idea)
    - » rsa-md5 means md5 from RSA!
  - check your latest documentation ...

in K5, one more major protocol change

- double encryption in K4 eliminated
  - e.g., TGS reply has service ticket encrypted by service key, encrypted with user key
  - in K5, basically concatenated together one after the other
- K5 uses string to key transformation but adds salt:
  - salt is complete principal name

K5, new ticket option

- forwardable ticket
  - user can ask for ticket to be sent to another host later
- renewable tickets
  - tickets have 2-tier lifetime scheme
  - standard lifetime and renewable lifetime
  - must be resubmitted to KDC in order for renewable in case of troubler
- postdated ticket
  - ticket that can be used later, useful for batch jobs

K5 - preauthentication

- K4 could have dictionary and brute-force attacks made against it
  - KDC gives ticket granting ticket for any principal in database to any client
  - offline attack can thus be made against any principal
- K5 makes more difficult with preauthentication feature
  - client must prove identity before getting ticket
- e.g., done by proving knowledge of shared key between client and KDC

misc issues: windows - practical use

- you can end up with single sign-on to "Active Directory"
- this will give you file shares
- printing
- some limited support for email depending on email clients?
- remember this is an authentication-oriented service
- uses HMAC-MD5 and RC4 for encryption as default, DES added later

UNIX implementation

- telnet/ftp may use it
  - telnet -x can even do encryption
- rsh/rlogin/rcp have used it
  - ironically made better as a consequence
- popper in Heimdal (pop server)
- don't assume encryption unless you know better
  - implementation dependent

cross-realm trust

- 2 or more domains shares the same encryption keys
- 2 principals created in each realm
  - trust may be 1-way, A trusts B, but not B trusts A
- cross-realm trust is N**2
  - may use shared realm to get around this
- of course more principals we have ... the less trust results

security and other considerations

- all apps should use it - few do
  - if one does not, the user password is exposed
  - it could be sniffed if mail app does not use it
- dependent on goodness/safeness of said user password
  - one hopes Alice's password is not Alice, password, or bob ...
- KDC may be a single point of failure
- security of KDC itself is VERY important
  - root compromise would be bad

security and other considerations

- Kerberos is single-user/per host system
  - keys may be stored in /tmp directory
- root compromise of client machine gives access to those keys
- are we still using DES with K5?
  - objectionable especially if encryption is actually used
- K4 may suffer from offline dictionary attacks

ports used by Kerberos

- K5 ticket service on 88 udp/tcp
- K5 kpassword service for client password changes
  - 749/TCP
- K5 to K4 ticket conversion, 4444/UDP
- K5 admin service (UNIX), 749/TCP
- Master/Admin KDC, 464/UDP (older password-changing protocol)
- K4 uses 750/751/761

study questions

- what pros/cons exist for putting the KDC on a windows box?
- what issues exist re user passwords and Kerberos?
- what issues exist re applications and Kerberos in terms of authentication/encryption?