


Network Security: Kerberos

Guevara Noubir
noubir@ccs.neu.edu
 Network Security

Reading assignment: Chapters 13-14


G. Noubir, Network Security Kerberos



Outline

- Kerberos V4
- Kerberos V5

G. Noubir, Network Security Kerberos 2



What is Kerberos?

- Initially, a secret key based service for authentication in a network ...
- Originally designed at MIT based on Needham-Schroeder work
- Components:
 - Key Distribution Center (KDC) running on a physically secure node
 - Library of subroutines used by distributed applications
- Environment:
 - Users connect on a workstation with username/password that are used by the workstation to access remote resources on behalf of the user
- Kerberized applications:
 - Telnet (RFC 854)
 - rtools (i.e., rlogin, rcp, rsh)
 - Network File System (NFS: RFC 1094)
 - Windows 2000, Windows Server 2003, ...

LDAP, CIFS/SMB remote file access, Common Internet File System (CIFS) is the new name of Microsoft's SMB protocol that is mainly used for file and print sharing, Secure dynamic DNS update, Distributed File System Management, Host to Host IPsec using ISAKMP, Secure intranet Web services using IIS, Authenticate certificate request to certification authority (CA), DCOM RPC security provider

G. Noubir, Network Security Kerberos 3

Tickets and Ticket-Granting Tickets

- **Principal:** each user and resource that will be using Kerberos
- KDC shares a **master key** with each principal
- When *A* wants to talk to *B* it requests a **ticket** from the KDC:
 - $Ticket = K_B(A, K_{AB})$
 - Session key (K_{AB}) and **ticket** are called *A*'s credentials to *B*
- A session key (S_A) is used for the session
- The KDC sends:
 - $K_A(S_A, TGT), TGT = K_{KDC}(A, S_A, Expiration-time)$ (TGT is ticket-granting ticket)
- In practice Ticket-Granting Servers (TGS) are the same as KDC and Authentication Servers (AS)

Configuration

- Each principal has its own secret key
- The KDC has a database of:
 - (name of principal, master key)
 - The master keys are stored in the database encrypted by **KDC master key**
 - Master keys of humans are derived from their password
- Old implementations use DES but Kerberos V5 has fields in packets for specifying the crypto algorithm and many implementations support 3DES, AES, RC4
- Why TGT?
 - The KDC doesn't need any volatile data, but only a large static database
 - Workstation does not need to keep the secret key

Logging into the Network

- Obtaining a session key and TGT:
 - *A* -> Workstation (WS): *A*, *password*
 - *WS* -> *KDC*: [AS_REQ] (Authentication Server Request)
 - *KDC* -> *WS*: [AS_REP] $K_A(S_A, TGT)$
- In Kerberos V4 the WS waits until it receives the reply before requesting the user's password
- In Kerberos V5 the WS proves that it has the user's key first. Why?

Accessing a Remote Node

Example: rlogin B

- A → WS: "rlogin B"
- WS → KDC: [TGS_REQ] (TGT, Authenticator)
 - Authenticator = $S_A(\text{timestamp})$
- KDC → WS: [TGS_REP] ($S_B(B, K_{AB})$, ticket-to-B)
 - Ticket-to-B = $K_B(A, K_{AB})$
- WS → B: [AP_REQ] (A, authenticator, ticket-to-B)
- B → WS: [AP_REP] ($K_{AB}(\text{timestamp}+1)$)
- Timestamps replay:
 - Timestamps within acceptable time skew (~ 5 minutes) are stored
 - Timestamps older than the skew are rejected
 - Doesn't help against replays with replicated services unless if server unique address is included
- Other security services can be used: integrity and/or encryption

Replicated KDCs

- A single KDC => single point of failure and performance bottleneck
 - Nodes can be down or network inaccessible because broken links
- Solution replicate the KDC: same master key, database
- One KDC holds the master copy
 - Any update is made to the master copy: add/modify/delete entry
 - Other sites periodically download the database on timeouts or human requests
- There is still a single point of failure problem but most operations are read-only
- KDC databases are downloaded unencrypted to replicas
 - Risks:
 - learn principals names and properties,
 - switch K_x by K_x avoided by using a cryptographic hash and a timestamp

REALMS

- Not everybody trusts the same KDC
 - There exists various competing companies and organizations
- A single replicated KDC requires:
 - Further trust, further security (one compromised replica leads to compromise of all principals)
- Principals are divided into **realms**:
 - A realm consists of a KDC (with its replicas)
- In V4 a principal has three components of upto 40 bytes
 - <NAME, INSTANCE, REALM>
 - E.g., a service NAME = *fileserv* for a fileserver, INSTANCE = machine on which the service is running
 - For human principals usually INSTANCE = NULL STRING, sometimes INSTANCE = role e.g., system-manager

Inter-Realm Authentication

- $A@KDC1$ wants to communicate with $B@KDC2$
 - $A \rightarrow KDC1$: [TGS_REQ]($A@KDC1$, $KDC2@KDC1$)
 - $KDC1 \rightarrow A$: credentials to $KDC2$
 - $A \rightarrow KDC2$: [TGS_REQ]($A@KDC1$, $B@KDC2$)
 - $KDC2 \rightarrow A$: credentials to B
 - $A \rightarrow B$: [AP_REQ]
- Only a one level inter-realm authentication chain is authorized in Kerberos V4
 - Goal: avoid allowing a realm to impersonate principals of other realms

Key Version Numbers

- If B changes its master key
 - Already issued tickets are no more valid and will fail
- If the KDC changes its master key TGT are no more valid
- First solution:
 - Attempts fail leading to requesting new tickets but
 - Inconvenient specially for batch processes
- Kerberos solution:
 - Include version number of keys in tickets and protocol messages
 - Network resources (including KDCs) remember previous keys (for about 21 hours = expiration time of tickets)
 - Too complex for human users => short instability period

Encryption for Privacy and Integrity

- How to provide simultaneous privacy and integrity
- Existing solutions:
 - CBC encryption and integrity with two different keys
 - CBC encryption of data concatenated with checksum
- Kerberos solution: Plaintext Cipher Block Chaining (PCBC)
 - XOR c_n with m_{n+1} and m_n and append known constant to plaintext
 - Modifying c_n during transmission results in errors in all subsequent messages
 - Flaws: e.g., swap of two adjacent blocks only impacts two blocks
- Kerberos initially used DES

Encryption for Integrity Only

- DES was assumed not to be fast enough in software implementations
- Kerberos V4 uses a checksum of the message concatenated with the session key
- The checksum algorithm was not documented
- Problems with Kerberos V4 checksum:
 - May be weaker than a cryptographically designed message digest
 - It might be possible to recover the session key from the plaintext and checksum
- Kerberos V5 allows the choice of a checksum algorithm (when combined with encryption) and message digests (for integrity only)

Network Layer Addresses

- Tickets include the network address (IPv4)
- Why?
 - Prevent A to give its token to a third party
 - Prevent an eavesdropper from using a stolen ticket
- Doesn't really improve security
 - It is easy to impersonate a network address
 - Prevents delegation which may be useful
 - Problems with NAT and migration to newer address formats

Message Formats: FIELDS

- NAME/INSTANCE/REALM: variable length null-terminated character strings
- TIMESTAMP: unix time representation 4 bytes, granularity: second, limited to 2038
- D BIT: Direction bit to prevent reflection of messages
 - Set to 1, when from higher IP address to lower IP address or higher port number to lower port number when used on the same machine
- LIFETIME: 1 byte, 5 minutes granularity => 21 hours
- 5-millisecond timestamp
- PADDING: up to 7 bytes
- NETWORK ADDRESS: 4 bytes IPv4
- SESSION KEY: 8 bytes (DES key)
- KEY VERSION NUMBER: 1 byte
- KERBEROS VERSION NUMBER: 1 byte
- MESSAGE TYPE: AS_REQ, AS_REPLY/TGS_REQ, AP_REQ/TGS_REQ, AP_REQ_MUTUAL, AS_ERR, PRIV (encrypted+integrity), SAFE (integrity), AP_ERR
- B BIT: big-endian (1) or little-endian (2)



Message Formats

- Data structures:
 - Tickets
 - Authenticators
 - Credentials
- Messages: 10 messages
 - AS_REQ, AS_REPLY/TGS_REP, AP_REQ/TGS_REQ, AP_REQ_MUTUAL, AS_ERR, PRIV (encrypted+integrity), SAFE (integrity), AP_ERR



Kerberos V5

- To allow flexibility in the choice of cryptographic algorithms Kerberos uses **ASN.1** (Abstract Syntax Notation)
 - Allows also various network address formats
 - However, it adds substantial overhead
- **NAMES:**
 - V4: (NAME, INSTANCE, REALM) -> V5: (NAME, REALM)
 - NAME: {type, sequence of strings}
 - V4: REALM = DNS name, but in V5 a REALM can also be X.500 names or other name types



V5: Delegation

- In V4 only way to delegate is to give the master key
 - Why?
- In V5 explicit delegation:
 - A can request a TGT to be used from a specified network address
 - It's a policy decision to accept such tickets by services
 - Explicit delegation allows auditing
 - Delegation can be restricted:
 - to a set of explicitly specified services
 - the TGT can include an AUTHORIZATION-DATA field that is application specific and can be interpreted by the application
 - Allowing delegation is optional: A *flag* TGT can indicate if delegation is authorized:
 - How does the KDC know? E.g., configured in the user's KDC entry
 - Forwardable Flag in the TGT => can be exchanged with a TGT for a different address
 - Proxiable Flag in the TGT => can request tickets for another address (e.g., Bob)

Tickets Lifetimes

- START-TIME, END-TIME, AUTHTIME, RENEW-TILL
- Unlimited lifetime
 - Format in ASN.1, can specify up to Dec. 31, 9999, granularity = 1 second
 - Problem of revoking long lived tickets!
- Renewable tickets:
 - A ticket valid for 100 years may have to be renewed every day (RENEW-TILL field)
 - Why? Simplifies revocation: KDC only needs to remember revoked tickets until their next due renewal time
- Postdated tickets
 - Goal: Issue a ticket valid for a time interval in the future
 - How: use START-TIME timestamp
 - Revocation: postdated tickets include an INVALID flag that has to be cleared by the KDC before use (unless if revoked by the creator)
 - Such tickets are marked as POSTDATED and could be rejected by an application
 - MAY-POSTDATE flag in TGT indicates if KDC can issue postdated tickets

Keys

- Key versions
 - Keys are stored in the KDC database as: $\langle key, p_kvno, k_kvno \rangle$
 - key is encrypted using the KDC master key
 - p_kvno : principal key version number
 - k_kvno : KDC key version number
 - In V4 the KDC only remembers current keys of principals
 - In V5 the KDC needs to remember old keys because ...
- Making master keys different in different realms
 - Human principals can use the same password on different realms
 - The password to key conversion uses a hash function and the realm name as one parameter
 - Why?

Cryptographic Algorithms

- Kerberos V5 allows insertion of different encryption algorithms
 - In practice still uses DES, 3DES, RC4, but also AES (RFC 3962, 2/2005)
- Repaired weaknesses:
 - Jueneman checksum and PCBC, HMAC
- Integrity-Only algorithms: 5 (three mandatory and two optional)
 - $rsa-md5-des$, $des-mac$, $des-mac-k$, $rsa-md4-des$, $rsa-md4-des-k$
 - [confounder = random 64 bits | MD5(confounder | message)] encrypted in CBC mode with IV = 0, and $K' = K_{AB} \oplus F0\ F0\ F0\ F0\ F0\ F0\ F0\ F0$
 - (RSA mentioned because RSADSI company owns rights to MD4 and MD5)
- Encryption for Privacy and Integrity
 - $des-cbc-crc$, $des-cbc-md4$, $des-cbc-md5$
 - [confounder = random 64 bits | checksum (32/128) | message | PAD] encrypt in CBC mode with IV = 0,



Hierarchy of Realms

- V4 allows only one level of inter-realm communication
- V5 allows multilevel inter-realm communication
 - To increase trust V5 includes a *TRANSITED* field that includes the list of all the realms that have been transited
- Realms are organized in hierarchies



Other Services

- Evading password-guessing attacks
 - Solution to attack 1: Require PREAUTHENTICATION-DATA to be sent with TGT request
 - Solution to attack 2: Prevent issuing tickets to communicate with humans (because key is derived from a password)
 - Doesn't prevent eavesdroppers from carrying offline password guessing
- Key inside authenticator
 - Goal: allow *A* to have two separate communications with *B*
 - How: *A* can choose another key K_{AB} and send it within the authenticator to *B*
- Double TGT authentication
 - *B* doesn't remember/have its master key (e.g., workstation representing user *B*)
 - *A* requests a ticket to *B* and provides the *KDC* with both *A*'s TGT and *B*'s TGT
 - The *KDC* generates a ticket to *B* encrypted with *B*'s session key
 - Application: remote display from an application into an XWINDOW server



PKINIT

- Goal:
 - Combine public-key technology with KDCs
 - Transparent to existing servers
- How?
 - Users get a TGT or ticket from the KDC using their public key and certificate



KDC Database

- *name*: principal's name
- *key*: principal's key
- *p_kvno*: principal's key version number
- *max_life*: maximum lifetime for tickets issued for principal
- *max_renewable_life*: maximum total lifetime for renewable tickets
- *k_kvno*: KDC key version number under which key is encrypted
- *expiration-time*: time when entry expires
- *mod_date*: time of last modification of entry
- *mod_name*: name of the principal who made the last modification
- *flags*: policy related e.g., allow forwardable, proxiable, postdated
- *Password_expiration*: time when password expires
- *last_pwd_change*: time when user changed password
- *last_success*: time of last successful user login (last AS_REQ with correct pre-authentication data)
