INTERNET-DRAFT
draft-ietf-cat-iakerb-08.txt
Updates: RFC 1510, 1964
September 2001

Jonathan Trostle
Cisco Systems
Michael Swift
University of WA
Bernard Aboba
Microsoft
Glen Zorn
Cisco Systems

Initial and Pass Through Authentication Using Kerberos V5 and the GSS-API (IAKERB)

<draft-ietf-cat-iakerb-08.txt>

Status of this Memo

1. Abstract

This document defines extensions to the Kerberos protocol specification (RFC 1510 [1]) and GSSAPI Kerberos protocol mechanism (RFC 1964 [2]) that enables a client to obtain Kerberos tickets for services where the KDC is not accessible to the client, but is accessible to the application server. Some common scenarios where lack of accessibility would occur are when the client does not have an IP address prior to authenticating to an access point, the client is unable to locate a KDC, or a KDC is behind a firewall. The document specifies two protocols to allow a client to exchange KDC

messages (which are GSS encapsulated) with an IAKERB proxy instead of a KDC.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119 [6].

3. Motivation

When authenticating using Kerberos V5, clients obtain tickets from a KDC and present them to services. This method of operation works well in many situations, but is not always applicable. The following is a list of some of the scenarios that this proposal addresses:

(1) The client must initially authenticate to an access point in order to gain full access to the network. Here the client may be unable to directly contact the KDC either because it does not have an IP address, or the access point packet filter does not allow the client to send packets to the Internet before it authenticates to the access point [8].

(2) A KDC is behind a firewall so the client will send Kerberos messages to the IAKERB proxy which will transmit the KDC request and reply messages between the client and the KDC. (The IAKERB proxy is a special type of Kerberos application server that also relays KDC request and reply messages between a client and the KDC).

4. Overview

This proposal specifies two protocols that address the above scenarios: the IAKERB proxy option and the IAKERB minimal messages option. In the IAKERB proxy option (see Figure 1) an application server called the IAKERB proxy acts as a protocol gateway and proxies Kerberos messages back and forth between the client and the KDC. The IAKERB proxy is also responsible for locating the KDC and may additionally perform other application proxy level functions such as auditing. A compliant IAKERB proxy MUST implement the IAKERB proxy protocol.

```
         Client <---------> IAKERB proxy <----------> KDC
```

Figure 1: IAKERB proxying

The second protocol is the minimal messages protocol which is based on user-user authentication [4]; this protocol is targetted at environments where the number of messages, prior to key establishment, needs to be minimized. In the normal minimal messages protocol, the client sends its ticket granting ticket (TGT) to the

IAKERB proxy (in a KRB_TKT_PUSH message) for the TGS case. The IAKERB proxy then sends a TGS_REQ to the KDC with the client's TGT in the additional tickets field of the TGS_REQ message. The returned ticket will list the client as the ticket's server principal, and will be encrypted with the session key from the client's TGT. The IAKERB proxy then uses this ticket to generate an AP request that is sent to the client (see Figure 2). Thus mutual authentication is accomplished with three messages between the client and the IAKERB proxy versus four or more (the difference is larger if crossrealm operations are involved).

Subsequent to mutual authentication and key establishment, the IAKERB proxy sends a ticket to the client (in a KRB_TKT_PUSH message). This ticket is created by the IAKERB proxy and contains the same fields as the original service ticket that the proxy sent in the AP_REQ message, except the client and server names are reversed and it is encrypted in a long term key known to the IAKERB proxy. Its purpose is to enable fast subsequent re-authentication by the client to the application server (using the conventional AP request AP reply exchange) for subsequent sessions. In addition to minimizing the number of messages, a secondary goal is to minimize the number of bytes transferred between the client and the IAKERB proxy prior to mutual authentication and key establishment. Therefore, the final service ticket (the reverse ticket) is sent after mutual authentication and key establishment is complete, rather than as part of the initial AP_REQ from the IAKERB proxy to the client. Thus protected application data (e.g., GSS signed and wrapped messages) can flow before this final message is sent.

The AS_REQ case for the minimal messages option is similar, where the client sends up the AS_REQ message and the IAKERB proxy forwards it to the KDC. The IAKERB proxy pulls the client TGT out of the AS_REP message; the protocol now proceeds as in the TGS_REQ case described above with the IAKERB proxy including the client's TGT in the additional tickets field of the TGS_REQ message.

A compliant IAKERB proxy MUST implement the IAKERB proxy protocol, and MAY implement the IAKERB minimal message protocol. In general, the existing Kerberos paradigm where clients contact the KDC to obtain service tickets should be preserved where possible.

For most IAKERB scenarios, such as when the client does not have an IP address, or cannot directly contact a KDC, the IAKERB proxy protocol should be adequate. If the client needs to obtain a crossrealm TGT (and the conventional Kerberos protocol cannot be used), then the IAKERB proxy protocol must be used. In a scenario where the client does not have a service ticket for the target server, it is crucial that the number of messages between the client and the target server be minimized (especially if the client and target server are in different realms), and/or it is crucial that the number of bytes transferred between the client and the target server be minimized, then the client should consider using the minimal messages protocol. The reader should see the security considerations section regarding the minimal messages protocol.

```
                Client  --------> IAKERB proxy
                        TKT_PUSH (w/ TGT)

                Client            IAKERB proxy --------------------> KDC
                                                TGS_REQ with client
                                                TGT as additional TGT

                Client            IAKERB proxy <-------------------- KDC
                                                TGS_REP with service
                                                ticket

                Client <--------  IAKERB proxy                       KDC
                         AP_REQ

                Client -------->  IAKERB proxy                       KDC
                         AP_REP

           -------------------------------------------------------------
            post-key establishment and application data flow phase:

                Client <--------  IAKERB proxy                       KDC
                        TKT_PUSH (w/ticket targetted at IAKERB proxy
                               to enable fast subsequent authentication)
```

Figure 2: IAKERB Minimal Messages Option: TGS case

5. GSSAPI Encapsulation

The mechanism ID for IAKERB proxy GSS-API Kerberos, in accordance with the mechanism proposed by SPNEGO [7] for negotiating protocol variations, is: {iso(1) org(3) dod(6) internet(1) security(5) mechanisms(5) iakerb(10) iakerbProxyProtocol(1)}. The proposed mechanism ID for IAKERB minimum messages GSS-API Kerberos, in accordance with the mechanism proposed by SPNEGO for negotiating protocol variations, is: {iso(1) org(3) dod(6) internet(1) security(5) mechanisms(5) iakerb(10) iakerbMinimumMessagesProtocol(2)}.

NOTE: An IAKERB implementation does not require SPNEGO in order to achieve interoperability with other IAKERB peers. Two IAKERB implementations may interoperate in the same way that any two peers can interoperate using a pre-established GSSAPI mechanism. The above OID's allow two SPNEGO peers to securely negotiate IAKERB from among a set of GSS mechanisms.

The AS request, AS reply, TGS request, and TGS reply messages are all

encapsulated using the format defined by RFC1964 [2]. This consists
of the GSS-API token framing defined in appendix B of [3]:

```
   InitialContextToken ::= [APPLICATION 0] IMPLICIT SEQUENCE {
           thisMech        MechType
                   -- MechType is OBJECT IDENTIFIER
                   -- representing iakerb proxy or iakerb min messages
           innerContextToken ANY DEFINED BY thisMech
                   -- contents mechanism-specific;
                   -- ASN.1 usage within innerContextToken
                   -- is not required
        }
```

The innerContextToken consists of a 2-byte TOK_ID field (defined below), followed by the Kerberos V5 KRB_AS_REQ, KRB_AS_REP, KRB_TGS_REQ, or KRB_TGS_REP messages, as appropriate. The TOK_ID field shall be one of the following values, to denote that the message is either a request to the KDC or a response from the KDC.

Message	TOK_ID
KRB_KDC_REQ	00 03
KRB_KDC_REP	01 03

We also define the token ID for the KRB_TKT_PUSH token (defined below and used in the minimal messages variation):

Message	TOK_ID
KRB_TKT_PUSH	02 03

For completeness, we list the other RFC 1964 defined token ID's here:

Message	TOK_ID
AP_REQ	01 00
AP_REP	02 00
KRB_ERROR	03 00

6. The IAKERB proxy protocol

The IAKERB proxy will proxy Kerberos KDC request, KDC reply, and KRB_ERROR messages back and forth between the client and the KDC as illustrated in Figure 1. Messages received from the client must first have the Kerberos GSS header (RFC1964 [2]) stripped off. The unencapsulated message will then be forwarded to a KDC. The IAKERB proxy is responsible for locating an appropriate KDC using the realm information in the KDC request message it received from the client. In addition, the IAKERB proxy SHOULD implement a retry algorithm for KDC requests over UDP (including selection of alternate KDC's if the initial KDC does not respond to its requests). For messages sent by the KDC, the IAKERB proxy encapsulates them with a Kerberos GSS header before sending them to the client.

We define two new Kerberos error codes that allow the proxy to indicate the following error conditions to the client:

(a) when the proxy is unable to obtain an IP address for a KDC in the client's realm, it sends the KRB_IAKERB_ERR_KDC_NOT_FOUND KRB_ERROR (80) message to the client.

(b) when the proxy has an IP address for a KDC in the client realm, but does not receive a response from any KDC in the realm (including in response to retries), it sends the KRB_IAKERB_ERR_KDC_NO_RESPONSE KRB_ERROR (81) message to the client.

To summarize, the sequence of steps for processing is as follows:

Servers:

1. For received KDC_REQ messages (with token ID 00 03)
   - process GSS framing (check OID)
     if the OID is not one of the two OID's specified in the GSSAPI Encapsulation section above, then process according to mechanism defined by that OID (if the OID is recognized). The processing is outside the scope of this specification. Otherwise, strip off GSS framing.
   - find KDC for specified realm (if KDC IP address cannot be obtained, send a KRB_ERROR message with error code KRB_IAKERB_ERR_KDC_NOT_FOUND to the client).
   - send to KDC (storing client IP address, port, and indication whether IAKERB proxy option or minimal messages option is being used)
   - retry with same or another KDC if no response is received. If the retries also fail, send an error message with error code KRB_IAKERB_ERR_KDC_NO_RESPONSE to the client.

2. For received KDC_REP messages
   - encapsulate with GSS framing, using token ID 01 03 and the OID that corresponds to the stored protocol option
   - send to client (using the stored client IP address and port)

3. For KRB_ERROR messages received from the KDC
   - encapsulate with GSS framing, using token ID 03 00 and the OID that corresponds to the stored protocol option
   - send to client (using the stored client IP address and port) (one possible exception is the KRB_ERR_RESPONSE_TOO_BIG error which can lead to a retry of the KDC_REQ message over the TCP transport by the server, instead of simply proxying the error to the client).

4. For sending/receiving AP_REQ and AP_REP messages
   - process per RFC 1510 and RFC 1964; the created AP_REP message
     SHOULD include the subkey (with same etype as the session key)
     to facilitate use with other key derivation algorithms outside
     of [2]. The subkey SHOULD be created using locally generated

entropy as one of the inputs (in addition to other inputs such as the session key).

Clients:

1. For sending KDC_REQ messages
   - create AS_REQ or TGS_REQ message
   - encapsulate with GSS framing (token ID 00 03 and OID corresponding to the protocol option).
   - send to server

2. For received KDC_REP messages
   - decapsulate by removing GSS framing (token ID 01 03)
   - process inner Kerberos message according to RFC 1510

3. For received KRB_ERROR messages
   - decapsulate by removing GSS framing (token ID 03 00)
   - process inner Kerberos message according to RFC 1510 and possibly retry the request (time skew errors lead to retries in most existing Kerberos implementations)

4. For sending/receiving AP_REQ and AP_REP messages
   - process per RFC 1510 and RFC 1964; the created AP_REQ message SHOULD include the subsession key in the authenticator field.

7. The IAKERB minimal messages protocol

The client MAY initiate the IAKERB minimal messages variation when the number of messages must be minimized (the most significant reduction in the number of messages can occur when the client and the IAKERB proxy are in different realms). SPNEGO [7] MAY be used to securely negotiate between the protocols (and amongst other GSS mechanism protocols). A compliant IAKERB server MAY support the IAKERB minimal messages protocol.

(a) AS_REQ case: (used when the client does not have a TGT)

We apply the Kerberos user-user authentication protocol [4] in this scenario (other work in this area includes the IETF work in progress effort to apply Kerberos user user authentication to DHCP authentication).

The client indicates that the minimal message sub-protocol will be used by using the appropriate OID as described above. The client sends the GSS encapsulated AS_REQ message to the IAKERB proxy, and the IAKERB proxy processes the GSS framing (as described above for the IAKERB proxy option) and forwards the AS_REQ message to the KDC.

The IAKERB proxy will either send a KRB_ERROR message back to the client, or it will send an initial context token consisting of the GSS header (minimal messages OID with a two byte token header 01 03), followed by an AS_REP message. The AS_REP message will contain the AP_REQ message in a padata field; the ticket in the AP_REQ is a user-user ticket encrypted in the session key from the client's original TGT. We define the padata type PA-AP-REQ with type number 25. The corresponding padata value is the AP_REQ message without any GSS framing. For the IAKERB minimal messages AS option, the AP_REQ message authenticator MUST include the RFC 1964 [2] checksum. The mutual-required and use-session-key flags are set in the ap-options field of the AP_REQ message.

The protocol is complete in the KRB_ERROR case (from the server perspective, but the client should retry depending on the error type). If the IAKERB proxy receives an AS_REP message from the KDC, the IAKERB proxy will then obtain the client's TGT from the AS_REP message. The IAKERB proxy then sends a TGS_REQ message with the client's TGT in the additional tickets field to the client's KDC (ENC-TKT-IN-SKEY option).

The IAKERB proxy MAY handle returned KRB_ERROR messages and retry the TGS request message (e.g. on a KRB_ERR_RESPONSE_TOO_BIG error, switching to TCP from UDP). Ultimately, the IAKERB proxy either proxies a KRB_ERROR message to the client (after adding the GSS framing), sends one of the new GSS framed KRB_ERROR messages defined above, or it receives the TGS_REP message from the KDC and then creates the AP_REQ message according to RFC 1964 [2]. The IAKERB proxy then sends a GSS token containing the AS_REP message with the AP_REQ message in the padata field as described above. (Note: although the server sends the context token with the AP_REQ, the client is the initiator.) The IAKERB proxy MUST set both the mutual-required and use-session-key flags in the AP_REQ message in order to cause the client to authenticate as well. The authenticator SHOULD include the subsession key (containing locally added entropy). The client will reply with the GSSAPI enscapsulated AP_REP message, if the IAKERB proxy's authentication succeeds (which SHOULD include the subkey field to facilitate use with other key derivation algorithms outside of [2]). If all goes well, then, in order to enable subsequent efficient client authentications, the IAKERB proxy will then send a final message of type KRB_TKT_PUSH containing a Kerberos ticket (the reverse ticket) that has the IAKERB client principal identifier in the client identifier field of the ticket and its own principal identity in the server identifier field of the ticket (see Figure 3):

```
KRB_TKT_PUSH :: = [APPLICATION 17] SEQUENCE {
  pvno[0]             INTEGER,  -- 5 (protocol version)
  msg-type[1]         INTEGER,  -- 17 (message type)
  ticket[2]           Ticket
}
```

NOTE: The KRB_TKT_PUSH message must be encoded using ASN.1 DER. The key used to encrypt the reverse ticket is a long term secret key chosen by the IAKERB proxy. The fields are identical to the AP_REQ ticket, except the client name will be switched with the server name, and the server realm will be switched with the client realm. (The one other exception is that addresses should not be copied from the AP_REQ ticket to the reverse ticket). Sending the reverse ticket allows the client to efficiently initiate subsequent reauthentication attempts with a RFC1964 AP_REQ message. Note that the TKT_PUSH message is sent after mutual authentication and key establishment are complete.

```
             Client  --------> IAKERB proxy --------------------> KDC
                     AS_REQ                  AS_REQ

             Client            IAKERB proxy <-------------------- KDC
                                             AS_REP w/ client TGT

             Client            IAKERB proxy --------------------> KDC
                                             TGS_REQ with client
                                             TGT as additional TGT

             Client            IAKERB proxy <-------------------- KDC
                                             TGS_REP with service
                                             ticket

             Client <--------  IAKERB proxy                       KDC
                      AS_REP w/ AP_REQ in padata field

             Client -------->  IAKERB proxy                       KDC
                      AP_REP

        -------------------------------------------------------------
         post-key establishment and application data flow phase:

             Client <--------  IAKERB proxy                       KDC
                     TKT_PUSH (w/ticket targetted at IAKERB proxy
                            to enable fast subsequent authentication)
```

Figure 3: IAKERB Minimal Messages Option: AS case

(b) TGS_REQ case: (used when the client has a TGT)

The client indicates that the minimal messages sub-protocol will be used by using the appropriate OID as described above. The client initially sends a KRB_TKT_PUSH message (with the GSS header) to the IAKERB proxy in order to send it a TGT. The IAKERB proxy will obtain the client's TGT from the KRB_TKT_PUSH message and then proceed to send a TGS_REQ message to a KDC where the realm of the KDC is equal to the realm from the server realm field in the TGT sent by the client in the KRB_TKT_PUSH message. NOTE: this realm could be the client's home realm, the proxy's realm, or an intermediate realm. The protocol then continues as in the minimal messages AS_REQ case described above (see Figure 2); the IAKERB proxy's TGS_REQ message contains the client's TGT in the additional tickets field (ENC-TKT-IN-SKEY option). The IAKERB proxy then receives the TGS_REP message from the KDC and then sends a RFC 1964 AP_REQ message to the client (with the MUTUAL AUTH flag set - see AS_REQ case).

To summarize, here are the steps for the minimal messages TGS protocol:

```
Client:
       (has TGT already for, or targetted at, realm X.ORG)
       sends TKT_PUSH message to server containing client's ticket
       for X.ORG (which could be a crossrealm TGT)

Server:
       (has TGT already targetted at realm X.ORG)
       sends to KDC (where KDC has principal id = server name,
         server realm from client ticket) a TGS_REQ:
       TGT in TGS_REQ is server's TGT
       Additional ticket in TGS_REQ is client's TGT from TKT_PUSH
         message
       Server name in TGS_REQ (optional by rfc1510) is not present
       Server realm in TGS_REQ is realm in server's TGT - X.ORG

KDC:
       Builds a ticket:
          Server name = client's name
          Client name = server's name, Client realm = server's realm
          Server realm = client's realm
          Encrypted with: session key from client's TGT (passed in
             additional tickets field)
       Build a TGS_REP
          Encrypted with session key from server's TGT
       Sends TGS_REP and ticket to server

Server:
       Decrypts TGS_REP from KDC using session key from its TGT
       Constructs AP_REQ
           Ticket = ticket from KDC (which was encrypted with
                    client's TGT session key)
           authenticator clientname = server's name (matches
             clientname in AP-REQ ticket)
           authenticator clientrealm = server's realm
           subsession key in authenticator is present (same
           etype as the etype of the session key in the ticket)
           checksum in authenticator is the RFC 1964 checksum
           sequence number in authenticator is present (RFC 1964)
           ap-options has both use-session-key and mutual-required
           flags set
       Sends AP_REQ (with GSS-API framing) to client

Client:
```

```
          Receives AP_REQ
          Decrypts ticket using session key from its TGT
          Verifies AP_REQ
          Builds AP_REP and sends to server (AP_REP SHOULD include
          subkey field to facilitate use with other key derivation
          algorithms outside of [2] e.g., [8] and its successors.
```

Some apps may have their own message protection key derivation algorithm and protected message format. AP_REP includes the sequence number per RFC 1964.)

Server:

Verifies AP-REP. Builds reverse ticket as described above and sends reverse ticket to client using the KRB_TKT_PUSH message. The reverse ticket is the same as the AP_REQ ticket except the client name, realm are switched with the server name, realm fields and it is encrypted in a secret key known to the IAKERB proxy.

8. Addresses in Tickets

In IAKERB, the machine sending requests to the KDC is the server and not the client. As a result, the client should not include its addresses in any KDC requests for two reasons. First, the KDC may reject the forwarded request as being from the wrong client. Second, in the case of initial authentication for a dial-up client, the client machine may not yet possess a network address. Hence, as allowed by RFC1510 [1], the addresses field of the AS and TGS requests SHOULD be blank and the caddr field of the ticket SHOULD similarly be left blank.

9. Security Considerations

Similar to other network access protocols, IAKERB allows an unauthenticated client (possibly outside the security perimeter of an organization) to send messages that are proxied to interior servers. When combined with DNS SRV RR's for KDC lookup, there is the possibility that an attacker can send an arbitrary message to an interior server. There are several aspects to note here:

(1) in many scenarios, compromise of the DNS lookup will require the attacker to already have access to the internal network. Thus the attacker would already be able to send arbitrary messages to interior servers. No new vulnerabilities are added in these scenarios.

(2) in a scenario where DNS SRV RR's are being used to locate the KDC, IAKERB is being used, and an external attacker can modify DNS responses to the IAKERB proxy, there are several countermeasures to prevent arbitrary messages from being sent to internal servers:

(a) KDC port numbers can be statically configured on the IAKERB proxy. In this case, the messages will always be sent to KDC's. For an organization that runs KDC's on a static port (usually port 88) and does not run any other servers on the same port, this countermeasure would be easy to administer and should be effective.

(b) the proxy can do application level sanity checking and filtering. This countermeasure should eliminate many of the above attacks.

(c) DNS security can be deployed. This countermeasure is probably overkill for this particular problem, but if an organization has already deployed DNS security for other reasons, then it might make sense to leverage it here. Note that Kerberos could be used to protect the DNS exchanges. The initial DNS SRV KDC lookup by the proxy will be unprotected, but an attack here is at most a denial of service (the initial lookup will be for the proxy's KDC to facilitate Kerberos protection of subsequent DNS exchanges between itself and the DNS server).

In the minimal messages protocol option, the application server sends an AP_REQ message to the client. The ticket in the AP_REQ message SHOULD NOT contain authorization data since some operating systems may allow the client to impersonate the server and increase its own privileges. If the ticket from the server connotes any authorization, then the minimal messages protocol should not be used. Also, the minimal messages protocol may facilitate denial of service attacks in some environments; to prevent these attacks, it may make sense for the minimal messages protocol server to only accept a KRB_TGT_PUSH message on a local network interface (to ensure that the message was not sent from a remote malicious host).

10. Acknowledgements

We thank the Kerberos Working Group chair, Doug Engert, for his efforts in helping to progress this specification. We also thank Ken Raeburn for his comments and the other working group participants for their input.

11. References

[1] J. Kohl, C. Neuman, "The Kerberos Network Authentication Service (V5)", RFC 1510.

[2] J. Linn, "The Kerberos Version 5 GSS-API Mechanism", RFC 1964.

[3] J. Linn, "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743.

[4] D. Davis, R. Swick, "Workstation Services and Kerberos Authentication at Project Athena", Technical Memorandum TM-424, MIT Laboratory for Computer Science, February 1990.

[5] S. Bradner, "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

[6] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[7] E. Baize, D. Pinkas, "The Simple and Protected GSS-API Negotiation Mechanism," RFC 2478, December 1998.

[8] Part 11: Wireless LAN Medium Access Control (MAC) and Physical
Layer (PHY) Specifications, ANSI/IEEE Std. 802.11, 1999 Edition.

12. Author's Addresses

Jonathan Trostle
Cisco Systems
170 W. Tasman Dr.
San Jose, CA 95134, U.S.A.
Email: jtrostle@cisco.com
Phone: (408) 527-6201

Michael Swift
University of Washington
Seattle, WA
Email: mikesw@cs.washington.edu

Bernard Aboba
Microsoft
One Microsoft Way
Redmond, Washington, 98052, U.S.A.
Email: bernarda@microsoft.com

Glen Zorn
Cisco Systems
Bellevue, WA U.S.A.
Email: gwz@cisco.com
Phone: (425) 468-0955

This draft expires on March 31st, 2002.