Network Working Group Jonathan Trostle
INTERNET-DRAFT Cisco Systems
Mike Swift
University of WA

The Lightweight Kerberos Protocol
<draft-trostle-lwkerb-01.txt>

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026 [6].

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet- Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This draft expires on November 30th, 2001. Please send comments to the authors.

1. Abstract

The Kerberos V5 protocol [3] allows network entities to authenticate and establish shared secret keys. Some network applications would benefit from a lightweight authentication mechanism with many of the benefits of Kerberos, but where the messages have fewer bytes than existing Kerberos messages. Also, we describe a protocol option that requires only two messages to be sent and received from the client, to support lightweight clients. This document describes a Kerberos-like protocol that does not use ASN.1 and is optimized for smaller messages. The protocol makes use of existing Kerberos infrastructure.

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 [2].

3. Protocol Overview

We define lw-ticket, lw-authenticator, and message constructs. We use the notation of TLS [5].

The current proposal does not eliminate the RFC 1510 Kerberos libraries from the client. The client obtains lw-tickets using conventional Kerberos exchanges with a KDC that also hosts the lw-KDC component. The client indicates its desire to obtain a lw-ticket, in addition to a conventional Kerberos ticket, by including a padata type in the KDC request message. The KDC returns the lw-ticket in a padata field of the KDC reply message.

Subsequent exchanges between the client and application server can use the lw-ap-req and lw-ap-rep messages in place of the Kerberos AP exchange. The advantage here is reduced processing and much smaller messages in the client server exchange. Our initial estimate is that there is approximately a 50% reduction in size of messages.

In addition, we define a new message that does not have a Kerberos analog: the lw-passthrough message. By using this option, the client does not have to contact the KDC at all (subsequent to obtaining its initial TGT). The lw passthrough option is valuable in environments where the client must minimize the messages it sends. To initiate it, the client sends a lw-ap-req to the application server, but the ticket in the lw-ap-req is either a TGT or a crossrealm TGT. The application server creates the lw-passthrough message by including the received lw-ap-req message and its own TGT and sending it to its local KDC. The local KDC, in case it cannot decrypt the lw-ap-req, then forwards it on to the next KDC, after replacing the ticket with its own crossrealm ticket targetted at the next KDC.

If the next KDC can decrypt the lw-ap-req, it validates it (except for authenticator time fields), and then re-encrypts the ticket with the session key from the accompanying ticket. It then sends the lw-passthrough message back to the previous KDC.

4. Protocol Constructs

We define lw-ap-req and lw-ap-rep messages that are sent by the initiator and responder, respectively. The lw-ap-rep is only sent by the target if the initiator has set the MUTUAL-AUTH flag.

```
lw-ap-req = { // version number will match tkt vno.
  uint16           message type
  lw-ticket        ticket
  lw-authenticator authenticator
}
```

```
lw-ap-reply = {
  uint16            version number
  uint16            message type
  uint16            extensions field length
  extfield          extensions field
```

```
        encryptedpart     lw-encreplypart
      }

      lw-encreplypart = {
        key            subkey
        uint64         sequence number
        uint32         time0
        uint32         time1
      }

      lw-ticket = {
        uint16          version number
        namestring      server name     // not present in service ticket
                                        // UTF-8 encoding
        namestring      server realm
        uint16          extensions field length
        extfield        extensions field
        encryptedpart   encticket
      }

      encticket = {
        namestring     client name
        namestring     client realm
        key            session key
        uint32         logon time      // not present in service ticket
        uint32         expiration time
        uint32         renew time      // not present in service ticket
      }

      namestring = {
        uint16         name length
        string         name           // UTF-8 encoding
      }

      extfield = {
        uint16         type0
        uint16         length0
        uchar[length0] data0
        uint16         type1
        uint16         length1
        uchar[length1] data1
        ...
      }

      lw-authenticator = {      //  version number matches ticket vno
        uint32      time0
        uint32      time1
        key         subkey
```

```
      uint64      sequence number
      uint16      extensions field length
      extfield    extensions field
    }
```

The lw-authenticator is encrypted using the session key from the ticket. Extension types 1-31 are reserved. We define the following extension field types for an authenticator:

```
  extension field type: 32
  extension field type 32 length: 4 (bytes)
  extension field type 32 data: context establishment flags bit
   vector (as in [4]):

  Delegation flag              1
  Mutual flag                  2
  Replay flag                  4
  Sequence flag                8
  Confidentiality flag         16
  Integrity flag               32
```

The bit vector is encoded in little-endian form. If this extension type is not present, then it is the same as sending type 32 with all of the above six bits set (delegation, mutual, replay, sequence, confidentiality, and integrity).

We define the following extension field types for a ticket:

```
    extension field type: 35
    extension field length: 8 bytes
    extension field data: both time fields from the authenticator.
```

This extension is used in the lw-passthrough message to allow an application server to quickly reject a message that is a replay or a clock skew error. Alternatively, the application server will learn about this problem after the lw-passthrough message is returned through multiple KDC's.

In the lwkerb passthrough option, the following ticket extension is placed into the ticket in the lw-ap-req by the user's KDC (the home KDC):

```
    extension field type: 36
    extension field data: the following servicetkt-skey structure
```

The servicetkt-skey structure is:

```
   servicetkt-skey = {
     encryptedpart   encsrvtktskey  // encrypted in the TGT skey
   }

   encsrvtktskey = {
     key             service tkt session key
   }
```

The structure is removed from the lw-ap-req and placed into the the lw-ap-reply by the lwkerb responder. The responder then sends the lw-ap-reply to the lwkerb initiator.

The following extension is placed into the lw-ap-req ticket extensions field by the lwkerb responder's KDC (the local KDC):

```
    extension field type: 37
    extension field data: the following enckdcreppart structure

   enckdcreppart = {
     encryptedpart   local-kdc-rep-body  // encrypted in srvtktskey
   }

   local-kdc-rep-body = {
     namestring server name  // UTF-8 encoding
     namestring server realm // UTF-8 encoding
     uint32     expiration time // could add to servicetkt-skey too
   }
```

This extension is removed from the lw-ap-req message and then placed in the lw-ap-reply extensions field by the lwkerb responder. The responder then sends the lw-ap-reply to the lwkerb initiator.

Here we define the key and encryptedpart structures:

```
   key = {
     uint16        keytype
     uint16        length
     uchar[length] keyvalue
   }

   encryptedpart = {
     uint16        etype
     uint16        keyversion
     uint16        length
     uchar[length] ciphertext    // define ciphertext for RC4,
                                 // AES etypes
   }
```

The encryption type is derived from the Kerberos encryption type.

We also define the following message that does not have a Kerberos analog; this message is used for the lwkerb passthrough option:

```
   lw-passthrough = {
     ap-req        lw-ap-req  // using client TGT or client
                              // xrealm TGT
     ticket        lw-ticket  // server or KDC lw-tgt
   }
```

Upon receiving such a message, a lw-KDC will check if the ap-req is targetted at itself. If so, it will validate that the authenticator in the ap-req decrypts successfully using the session key from the lw-ticket. The ticket is also decrypted, and the expiration is checked. The lw-KDC will NOT validate the time fields in the authenticator to check for replays. If all goes well, the lw-KDC will re-encrypt the ticket using the session key from the ticket in the lw-passthrough message and send the lw-passthrough message back to the entity it received it from.

If the ap-req is not targetted at the current lw-KDC, then the lw-KDC forwards it to the next lw-KDC after replacing the ticket with its own lw-tgt targetted at the next KDC. The old ticket is cached and used when the lw-KDC receives the lw-passthrough message back from the next KDC.

```
ErrorMessage = {
  uint16        error code
  uint16        extensions field length
  extfield      extensions field
}
```

The following error codes are reused from [3]:

```
lwapreqerr-bad-integrity     31
lwapreqerr-tkt-expired       32
lwapreqerr-repeat            34
lwapreqerr-not-us            35
lwapreqerr-badmatch          36
lwapreqerr-skew              37
```

along with errors 39-50, and 60 from [3].

5. Acknowledgements

The authors thank Doug Engert and Hannes Tschofenig for their feedback on this document.

6. Security Considerations

The entire draft discusses security.

7. References

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

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

[3] J. Kohl, C. Neuman. The Kerberos Network Authentication Service (V5), Request for Comments 1510.

[4] Linn, J., "The Kerberos V5 GSSAPI Mechanism", RFC 1964.

[5] Dierks T., Allen C. "The TLS Protocol, Version 1.0",

RFC 2246.

8. Expiration Date

This draft expires on November 30th, 2001.

9. Authors' Addresses

Jonathan Trostle
Cisco Systems
170 W. Tasman Dr.
San Jose, CA 95134
Email: jtrostle@cisco.com

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

10. Full Copyright Statement