Delay Tolerant Networking Research Group   S. Farrell
Internet Draft   Trinity College Dublin
Intended Status: Experimental   M. Ramadas
<draft-irtf-dtnrg-ltp-extensions-05.txt>   Ohio University
April 2007   S. Burleigh
Expires October 2007   NASA/Jet Propulsion Laboratory

Licklider Transmission Protocol - Extensions

Status of this Memo

Copyright Notice

Abstract

In an Interplanetary Internet setting deploying the Bundle protocol being developed by the Delay Tolerant Networking Research Group, the Licklider Transmission Protocol (LTP), is intended to serve as a reliable convergence layer over single hop deep-space RF links. LTP does ARQ of data transmissions by soliciting selective-acknowledgment

reception reports. It is stateful and has no negotiation or handshakes.

LTP is designed to provide retransmission-based reliability over links characterized by extremely long message round-trip times (RTTs) and/or frequent interruptions in connectivity. Since communication across interplanetary space is the most prominent example of this sort of environment, LTP is principally aimed at supporting "long-haul" reliable transmission in interplanetary space, but has applications in other environments as well.

This document describes extensions to LTP, and is part of a series of related documents describing LTP. Other documents in this series cover the motivation for LTP and the main protocol specification. We recommend reading all the documents in the series before writing code based on this document.

This document is a product of the Delay Tolerant Networking Research Group and has been reviewed by that group. No objections to its publication as an RFC were raised.

Table of Contents

1. Introduction

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 [B97].

Discussions on this internet-draft are being made in the Delay Tolerant Networking Research Group (DTNRG) mailing list. More information can be found in the DTNRG web-site at http://www.dtnrg.org This document describes extensions to the base LTP protocol [LTPSPEC]. The background to LTP is described in the

"motivation" document [LTPMOTIVE]. All the extensions defined in this document provide additional security features for LTP.

2. Security Extensions

The syntactical layout of the extensions are defined in Section 3.1.4 of the base protocol specification [LTPSPEC].

Implementers should note that the LTP extension mechanism allows for multiple occurrences of any extension tag, in both (or either) the header or trailer. For example, the LTP authentication mechanism defined below requires both header and trailer extensions, which both use the same tag.

2.1 LTP Authentication

The LTP Authentication mechanism provides cryptographic authentication of the segment.

Implementations MAY support this extension field. If they do not support this header then they MUST ignore it.

The LTP authentication extension field has the extension tag value 0x00.

LTP authentication requires three new fields, the first two of which are carried as the value of the extensions field of the LTP segment header, and the third of which is carried in the segment trailer.

The fields which are carried in the header extensions field are catenated together to form the extension value (with the leftmost octet representing the ciphersuite and the remaining octets the KeyID). The KeyID field is optional, and is determined to be absent if the extension value consists of a single octet.

Ciphersuite: an eight bit integer value with values defined below.

KeyID: An optional key identifier, the interpretation of which is out of scope for this specification (that is, implementers MUST treat these KeyID fields as raw octets, even if they contained an ASN.1 DER encoding of an X.509 IssuerSerial construct [PKIXPROF], for example).

The LTP-auth header extension MUST be present in the first segment from any LTP session which uses LTP authentication, but MAY be omitted from subsequent segments in that session. To guard against additional problems arising from lost segments, implementations SHOULD, where bandwidth allows, include these fields in a number of

segments in the LTP session. If the first segment (or any part thereof) is re-transmitted, then the LTP-auth header extension MUST be included in the re-transmission.

The field carried as a trailer extension is the AuthVal field. It contains the authentication value, which is either a message authentication code (MAC) or a digital signature. This is itself a structured field whose length and formatting depends on the ciphersuite.

If for some reason the sender includes two instances of LTP auth headers then there is a potential problem for the receiver in that presumably at least one of the AuthVal fields will not verify. There are very few situations where it would make sense to include more than one LTP auth extension in a single segment, since LTP is a peer-to-peer protocol. If however, keys are being upgraded then the sender might protect the segment with both the new and old keys. In such cases the receiver MUST search and can consider the LTP authentication valid so long as one AuthVal is correct.

For all ciphersuites, the input to the calculation is the entire encoded segment including the AuthVal extension tag and length, but not of course, including the AuthVal value.

We define three ciphersuites in this specification. Our approach is to follow the precedent set by TLS [TLS], and to "hardcode" all algorithm options in a single ciphersuite number. This means that there are 256 potential ciphersuites supported by this version of LTP-auth.

Ciphersuite	Value
OriginAuth	0
Signature	1
NULL	255

1. OriginAuth Ciphersuite

The OriginAuth ciphersuite involves generating a MAC over the LTP segment and appending the resulting AuthVal field to the end of the segment. There is only one MACing algorithm defined for this which is HMAC-SHA1-80 [HMAC]. The AuthVal field in this case contains just the output of the HMAC-SHA1-80 algorithm which is a fixed width field (10 octets).

2. Signature Ciphersuite

The Signature ciphersuite involves generating a digital signature of the LTP segment and appending the resulting AuthVal field to the end of the segment. There is only one signature algorithm currently defined for this which is RSA with SHA256 [RSA]. The AuthVal field in this case is simply the signature value, where the signature value occupies the minimum number of octets, e.g. 128 octets for a 1024 bit signature).

3. NULL Ciphersuite

The NULL ciphersuite is basically the same as the OriginAuth ciphersuite, but with a hardcoded key. This ciphersuite effectively provides only data integrity without authentication, and thus is subject to active attacks and is the equivalent of providing a CRC.

The hardcoded key to be used with this ciphersuite is the following:

```
   HMAC_KEY     :  c37b7e64 92584340
                :  bed12207 80894115
                :  5068f738
```

(The above is the test vector from RFC 3537 [WRAP].)

In each case the bytes which are input to the cryptographic algorithm consist of the entire LTP segment except the AuthVal. In particular the header extensions field which may contain the ciphersuite number and the KeyID field are part of the input.

The output bytes of the cryptographic operation are the payload of the AuthVal field.

The following shows an example LTP-auth header, starting from and including the extensions field

```
 ext  tag  sdnv  c-s  k-id
+----+----+----+----+----+
|0x11|0x00|0x02|0x00|0x24|
+----+----+----+----+----+
```

The Extensions field has the value 0x11 with the most-significant and least-significant nibble value 1, indicating the presence of one header and one trailer extension respectively. The next octet is the extension tag (0x00 for LTP-auth), followed by the SDNV encoded length of the ensuing data : a one-octet ciphersuite (0x00 meaning OriginAuth) and the KeyID (in this case with a short value of 0x24). The trailer extension (not shown above) should contain the AuthVal).

2.2 A Cookie mechanism

The use of cookies is a well known way to make denial-of-service attacks harder to mount. We define the cookie extension for use in environments where an LTP implementation is liable to such attacks.

The cookie is placed in a header extension field, and has no related trailer extension field. It has the extension tag value 0x01.

The cookie value can essentially be viewed as a sufficiently long random number, where the length can be determined by the implementation (longer cookies are harder to guess and therefore better, though using more bandwidth). Note that cookie values can be derived using lots of different schemes so long as they produce random looking and hard to guess values.

The first cookie inserted into a segment for this session is called the initial cookie.

Note that cookies do not outlast an LTP session.

The basic mode of operation is that an LTP engine can include a cookie in a segment at any time. After that time all segments corresponding to that LTP session MUST contain a good cookie value - that is, all segments both to and from the engine MUST contain a good cookie. Clearly, there will be some delay before the cookie is seen in incoming segments - implementations MUST determine an acceptable delay for these cases, and MUST only accept segments without a cookie until that time.

The cookie value can be extended at any time by catenating more random bits. This allows both LTP engines to contribute to the randomness of the cookie, where that is useful. It also allows a node which considers the cookie value too short (say due to changing circumstances) to add additional security. In this case, the extended cookie value becomes the "to-be-checked-against" cookie value for all future segments (modulo the communications delay as above).

It can happen that both sides emit segments containing an initial cookie before their peer has a chance to see any cookie. In that case two cookie extension fields MUST be included in all segments subsequently (once the traffic has caught up). That is, the sender and recipient cookies are handled independently. In such cases, both cookie values MUST be "good" at all relevant times (i.e. modulo the delay). In this case, the peer's initial cookie MUST arrive before the calculated delay for receipt of segments containing this engine's cookie - there is only a finite window during which a second cookie

can be inserted into the session.

A "good" cookie is therefore one which starts with the currently stored cookie value, or else a new cookie where none has been seen in that session so far. Once a cookie value is seen and treated as "good" (e.g. an extended value), the previous value is no longer "good".

Modulo the communications delay, segments with an incorrect or missing cookie value MUST be silently discarded.

If a segment is to be re-transmitted, (e.g. as a result of a timer expiring) then it needs to contain the correct cookie value at the time of (re-)transmission. Note that this may differ from what was the correct cookie value at the time of the original transmission.

3. Security Considerations

While there are currently some concerns about using the SHA-1 algorithm, these appear to only make it easier to find collisions. In that case, the use of HMAC with SHA-1 can still be considered safe. However, we have changed to use SHA-256 for the signature ciphersuite.

4. IANA Considerations

At present there are none known.

5. Acknowledgments

Many thanks to Tim Ray, Vint Cerf, Bob Durst, Kevin Fall, Adrian Hooke, Keith Scott, Leigh Torgerson, Eric Travis, and Howie Weiss for their thoughts on this protocol and its role in Delay-Tolerant Networking architecture.

Part of the research described in this document was carried out at the Jet Propulsion laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. This work was performed under DOD Contract DAA-B07-00-CC201, DARPA AO H912; JPL Task Plan No. 80-5045, DARPA AO H870; and NASA Contract NAS7-1407.

Thanks are also due to Shawn Ostermann, Hans Kruse, and Dovel Myers at Ohio University for their suggestions and advice in making various design decisions.

Part of this work was carried out at Trinity College Dublin as part of the Dev-SeNDT contract funded by Enterprise Ireland's technology

development programme.

6. References

6.1 Normative References

[B97] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[HMAC] Krawczyk, H. et al, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[LTPSPEC] Ramadas, M., Burleigh, S., and Farrell, S., "Licklider Transmission Protocol - Specification", draft-irtf-dtnrg-ltp-06.txt (Work in Progress), April 2007.

[RSA] Kaliski, B, Staddon J, "PKCS #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.

6.2 Informative References

[LTPMOTIVE] Burleigh, S., Ramadas, M., and Farrell, S., "Licklider Transmission Protocol - Motivation", draft-irtf-dtnrg-ltp-motivation-04.txt (Work in Progress), April 2007.

[PKIXPROF] Housley, R. et al, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[TLS] Dierks, T., Allen, C. "The TLS Protocol - Version 1.1", RFC 4346, April 2006.

[WRAP] Schaad, J. Housley, R. "Wrapping a Hashed Message Authentication Code (HMAC) key with a Triple-Data Encryption Standard (DES) Key or an Advanced Encryption Standard (AES) Key", RFC 3537, May 2003.

7. Author's Addresses

Stephen Farrell
Distributed Systems Group
Computer Science Department
Trinity College Dublin
Ireland
Telephone +353-1-608-3070
Email stephen.farrell@cs.tcd.ie

Manikantan Ramadas

Internetworking Research Group
301 Stocker Center
Ohio University
Athens, OH 45701
Telephone +1 (740) 593-1562
Email mramadas@irg.cs.ohiou.edu

Scott C. Burleigh
Jet Propulsion Laboratory
4800 Oak Grove Drive
M/S: 301-485B
Pasadena, CA 91109-8099
Telephone +1 (818) 393-3353
FAX +1 (818) 354-1075
Email Scott.Burleigh@jpl.nasa.gov

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.