Internet Draft
Document: draft-ietf-tls-delegation-01.txt

K. Jackson
LBNL
S. Tuecke
D. Engert
ANL

Expires: July 2001

Feburary 2002

TLS Delegation Protocol

Status of this Memo

Abstract

This document specifies a delegation protocol for use with the Transport Layer Security (TLS) protocol. When the TLS session is using X.509 certificates for authentication, then the delegation is of an X.509 Proxy Certificate, as defined in draft-ggf-x509-proxy. When the TSL session is using Kerberos 5 for authentication, then the delegation is of a Kerberos 5 forwardable ticket, as defined in RFC 1510.

Table of Contents

TLS Delegation Protocol.....1
Status of this Memo.....1
Abstract.....1

Table of Contents.....2
1. Introduction.....3
2. Assumed Session Properties.....3
3. TLS Record Layer.....3
4. Delegation Protocol.....4
4.1. Delegation Protocol Overview.....4
4.2. Delegation Protocol Messages.....5
4.3. Delegation Init.....5
4.4. Delegation Begin.....6
4.5. Credential Request.....6
4.6. Delegation Complete.....7
4.7. Delegation Error.....7
4.7.1. No Delegation.....8
4.7.2. Unsupported Credential Type.....8
4.7.3. Unsupported Version.....8
4.7.4. Delegation Denied.....8
4.7.5. Invalid Session.....9
5. Security Considerations.....9
6. References.....9
7. Acknowledgments.....9
8. Contact Information.....9

1. Introduction

The TLS protocol, as defined in RFC 2246 [2], provides for authentication and data protection for communication between two entities. However, missing from the protocol is a way to perform delegation of a credential.

This document defines extensions to the TLS protocol to allow it to perform delegation of a Proxy credential. When X.509 public key certificates are used, this delegation protocol will delegate a Proxy Certificate, as defined by draft-ggf-x509-proxy [5]. When Kerberos 5 is used, this delegation protocol will delegate a Kerberos 5 forwardable ticket, as defined by RFC 1510 [3].

The motivation for proxies is discussed at length in draft-ggf-x509-proxy, and therefore will not be discussed here.

Section 2 contains the assumptions this protocol makes about the underlying session established by TLS. Section 3 contains information about the necessary additions to the TLS Record Layer. Section 4 contains a description of the proposed delegation protocol.

Section 5 discusses security considerations relating to this TLS

Delegation Protocol. Section 6 contains the references. Section 7 contains acknowledgements. Section 8 contains contact information for the authors.

This document deals with the formatting of data in an external representation. The presentation language defined in RFC 2246 [2] will be used.

This document was written under the auspices of the Global Grid Forum Security Working Group. For more information on this and other related work, see http://www.gridforum.org/security.

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 RFC-2119 [1].

2. Assumed Session Properties

This protocol relies on several properties of the underlying TLS session. In particular the protocol assumes that the communications channel is integrity checked, protects against man-in-the-middle attacks, and prevents replay attacks. In the case of PKI based delegation, the delegatee must have authenticated itself to the delegator. It is a fatal protocol error to attempt to initiate the delegation protocol in a session that does not meet these minimum requirements.

3. TLS Record Layer

The TLS Protocol is a layered protocol. The TLS Record Protocol is the lowest layer. It provides the low-level message fragmentation, encryption/decryption, etc. Layered on top of this protocol are four record protocol clients: the handshake protocol, the alert protocol, the change cipher spec protocol, and the application data protocol. The delegation protocol is a fifth record protocol client. To support this the ContentType definition from section A.1 of RFC 2246 becomes:

```
enum { change_cipher_spec(20), alert(21),
       handshake(22), application_data(23),
       delegation(24), (255)
} ContentType;
```

4. Delegation Protocol

The TLS Delegation Protocol is a higher-level client of the TLS Record Protocol. This protocol is used to allow the initiator to

delegate the ability to act on its behalf to the acceptor in the event that the acceptor must make a request to a third party on behalf of the initiator. In the case of X.509 certificate based delegation, the delegated credential will be a Proxy Certificate as defined by draft-ggf-x509-proxy. In the case of Kerberos 5 based delegation, the delegated credential will be a Kerberos 5 forwardable ticket. Delegation request messages are supplied to the TLS Record Layer, where they are encapsulated within one or more TLSCiphertext structures, which are processed and transmitted as specified by the current active session state.

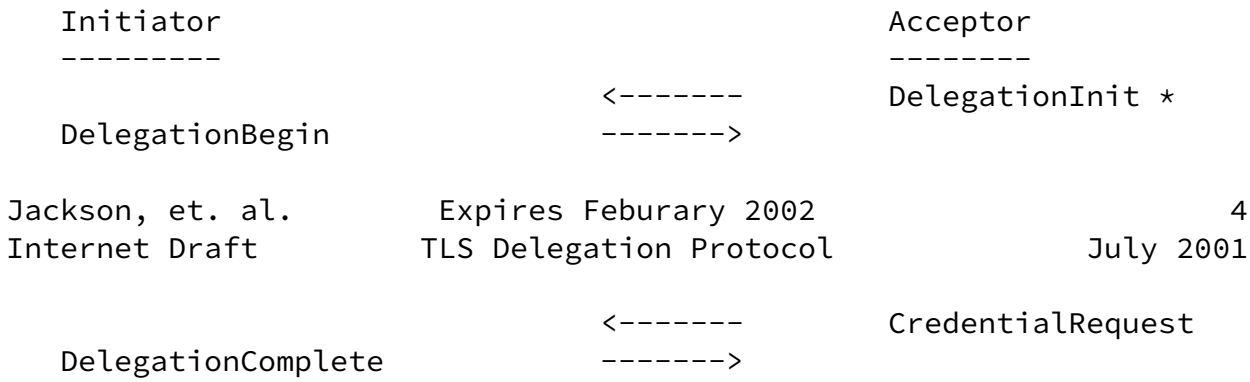
4.1. Delegation Protocol Overview

The TLS Delegation Protocol involves the following steps:

* Exchange delegation initiation messages and negotiate the credential type.

* Create the delegated credential.

Either the initiator or the acceptor may initiate the protocol. The acceptor optionally sends a DelegationInit message if it is initiating the protocol. The initiator sends a DelegationBegin message in response, or this message may be sent to initiate the protocol. The acceptor will now create an algorithm specific delegation request. It then sends a CredentialRequest message to the initiator containing the request. The initiator returns the resulting delegation credential to the acceptor in a DelegationComplete message. At this point, the delegation protocol is complete and the acceptor is in possession of a delegated credential. (See flow chart below.)



* Indicates optional or situation-dependent messages that are not always sent.

4.2. Delegation Protocol Messages

```
enum {
   delegation_init(8), delegation_begin(16),
   delegation_request(24), delegation_complete(32),
```

```
      delegation_error(40), (255)
   } DelegationType;

   struct {
      DelegationType msg_type;    /* delegation type */
      uint24 length;              /* bytes in message */
      select (DelegationType) {
         case delegation_init:         DelegationInit;
         case delegation_begin:        DelegationBegin;
         case credential_request:      CredentialRequest;
         case delegation_complete:     DelegationComplete;
         case delegation_error:        DelegationError;
      } body;
   } Delegation;
```

4.3. Delegation Init

When this message will be sent:
The acceptor may send the delegation request message at any time.

Meaning of this message:
Delegation init is a notification to the initiator that the acceptor would like the initiator to initiate the delegation protocol. The initiator may ignore the message if it is unwilling to perform delegation at this time, or it may respond with a no delegation error message if unwilling to perform delegation. This message will contain the type of requested delegation credential. This allows for the ability to delegate a credential of a different type than was authenticated with. This may be useful if crossing authentication boundaries.

After sending the delegation init message, acceptors should not repeat the request until the delegation protocol is complete.

Structure of this message:

```
      enum {
         PKI(8), Kerberos(9),
         (255)
      } CredentialType;

      struct {
         opaque policy_list<0..??>  [Not sure on the total size here]
      } PolicyList;
```



```
      struct {
         CredentialType requested_type;
         PolicyList policy_list;
```

```
   } DelegationInit;
```

requested_type
   The type of delegation credential the acceptor is requesting.
   Currently supported types are PKI and Kerberos.

policy_list
   An opaque structure that may contain policy information to be
   used by the initiator.

4.4. Delegation Begin

When this message will be sent:
   The initiator may send this message at any time to initiate the
   protocol, or it will send this message in response to a
   delegation begin message.

Structure of this message:

```
   struct {
      uint8 major, minor;
   } ProtocolVersion;

   struct {
      opaque policy_list<0..??>  [Not sure on the total size here]
   } PolicyList;

   struct {
      ProtocolVersion initiator_version;
      CredentialType type;
      PolicyList policy_list;
   } DelegationBegin;
```

initiator_version
   The version of the delegation protocol the initiator wishes to
   use. This should be the latest (highest valued) version
   supported by the initiator.

type
   The type of delegation credential the initiator is willing to
   delegate. Currently supported types are PKI and Kerberos 5.

policy_list
   An opaque structure that may contain policy information to be
   used by the acceptor when creating the credential request.

4.5. Credential Request

When this message will be sent:

The acceptor will send this message in response to the delegation begin message.

Structure of this message:

```
struct {
   ProtocolVersion acceptor_version;
   PolicyList policy_list;
   select (CredentialType) {
      case PKI:
         opaque CertReqMessage<1..?>;[Not sure of size]
      case Kerberos:
         opaque [Not sure what should go here, Doug?]
   };
} CredentialRequest;
```

acceptor_version
The version of the delegation protocol the acceptor is using. This should be the latest (highest valued) version supported by both the initiator and acceptor.

policy_list
An opaque structure that may contain policy information to be used by the acceptor when creating the credential request.

CertReqMessage
A certificate request message as defined in draft-ietf-pkix-rfc2511bis-00.txt [4].

4.6. Delegation Complete

When this message will be sent:
The initiator will send this message in response to the credential request message.

Structure of this message:

```
struct {
   select (CredentialType) {
      case PKI:
         opaque ASN.1Cert<0..2^24-1>;
      case Kerberos:
         opaque [I don't know what would go here, Doug?]
   };
} DelegationComplete;
```

ASN.1Cert
An X.509 Proxy Certificate.

4.7. Delegation Error

```
   enum {
      no_delegation(8), unsupported_credential_type(16),
      unsupported_version(24), invalid_session(32), (255)
   } DelegationErrorType;
```



```
   struct {
      DelegationErroryType msg_type;
      select (DelegationErrorType) {
         case no_delegation:                NoDelegation;
         case unsupported_credential_type   UnsupportedCredential;
         case unsupported_version           UnsupportedVersion;
         case delegation_denied             DelegationDenied;
         case invalid_session               InvalidSession;
      };
   } DelegationError;
```

4.7.1. No Delegation

When this message will be sent:
The initiator, in response to a delegation init message, may send this if unwilling to delegate to this acceptor.

Structure of this message:

```
struct { } NoDelegation;
```

4.7.2. Unsupported Credential Type

When this message will be sent:
The initiator, in response to a delegation init message, may send this if it does not support the credential type the acceptor requested. The acceptor, in response to a delegation begin message, may send this if it does not support the credential type the initiator is willing to delegate.

Structure of this message:

```
struct { } UnsupportedCredential;
```

4.7.3. Unsupported Version

When this message will be sent:
The acceptor, in response to a delegation begin message, may send this message if it is unable to support the protocol version the initiator requested. The initiator, in response to a delegation request message, may send this if unable to support the protocol version the acceptor requested.

Structure of this message:
struct {} UnsupportedVersion;

4.7.4. Delegation Denied

When this message will be sent:
The initiator, in response to a credential request message, may send this if it is unwilling to generate the requested credential.

Structure of this message:
struct { } DelegationDenied;

4.7.5. Invalid Session

When this message will be sent:
Either the initiator or acceptor must send this message if it receives a delegation message while the current session does not support the minimum requirements of the protocol.

Structure of this message:
struct { } InvalidSession;

5. Security Considerations

Security issues are discussed throughout this memo.

6. References

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

[2] Dierks, T. and C. Allen, "The TLS Protocol, Version 1.0," RFC 2246, January 1999.

[3] Kohl, J. and C. Neuman, "The Kerberos Network Authentication Service (V5)," RFC 1510, September 1993.

[4] Myers, M., C. Adams, D. Solo, and D. Kemp, "Certificate Request Message Format (CRMF)," Internet Draft draft-ietf-pkix-rfc2511bis-00.txt, November 2000.

[5] Tuecke, S., D. Engert, and M. Thompson, "Internet X.509 Public Key Infrastructure Proxy Certificate Profile," Internet Draft draft-ggf-x509-Proxy-05.txt, February 2001.

7. Acknowledgments

We are grateful to numerous colleagues for discussions on the topics covered in this paper, in particular (in alphabetical order, with apologies to anybody we've missed): Joe Bester, Randy Butler, Olivier Chevassut, William Johnston, Carl Kessleman, Ian Foster, Marty Humphrey, Clifford Neuman, Mary Thompson, Gene Tsudik, and Von Welch.

This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38 and under Contract DE-AC03-76SF00098 with the University of California; by the Defense Advanced Research Projects Agency under contract N66001-96-C-8523; by the National Science Foundation; and by the NASA Information Power Grid project.

8. Contact Information

Keith R. Jackson
Lawrence Berkeley National Laboratory
Berkeley, CA 94702
Phone: 510-486-4401
Email: KRJackson@lbl.gov

Steven Tuecke
Distributed Systems Laboratory
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
Email: tuecke@mcs.anl.gov

Doug Engert
Argonne National Laboratory
Argonne, IL 60439
Email: deengert@anl.gov