Network Working Group   V. Marinov
Internet-Draft   J. Schoenwaelder
Intended status: Standards Track   Jacobs University Bremen
Expires: August 27, 2007   February 23, 2007

Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP)
draft-marinov-isms-tlstm-00.txt

Status of this Memo

Copyright Notice

Abstract

This memo defines a Transport Model for the Simple Network Management Protocol which utilizes the Transport Layer Security (TLS) security protocol and the X.509 key management infrastructure.

Table of Contents

1. Introduction

This memo describes a Transport Model for the Simple Network Management Protocol, using the Transport Layer Security protocol [RFC4346] within a transport subsystem [I-D.ietf-isms-tmsm]. The transport model specified in this memo is referred to as the Transport Layer Security Transport Model (TLSTM).

This memo also defines a portion of the Management Information Base (MIB) for use with network management protocols in TCP/IP based internets. In particular it defines objects for monitoring and managing the TLS Transport Model for SNMP.

It is important to understand the SNMP architecture and the terminology of the architecture to understand where the Transport Model described in this memo fits into the architecture and how it interacts with other subsystems within the architecture.

1.1. The Internet-Standard Management Framework

For a detailed overview of the documents that describe the current Internet-Standard Management Framework, please refer to section 7 of RFC 3410 [RFC3410].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. MIB objects are generally accessed through the Simple Network Management Protocol (SNMP). Objects in the MIB are defined using the mechanisms defined in the Structure of Management Information (SMI). This memo specifies a MIB module that is compliant to the SMIv2, which is described in STD 58, RFC 2578 [RFC2578], STD 58, RFC 2579 [RFC2579] and STD 58, RFC 2580 [RFC2580].

1.2. Conventions

The terms "manager" and "agent" are not used in this document, because in the RFC 3411 [RFC3411] architecture, all SNMP entities have the capability of acting as either manager or agent or both depending on the SNMP applications included in the engine. Where distinction is required, the application names of Command Generator, Command Responder, Notification Originator, Notification Receiver, and Proxy Forwarder are used. See "SNMP Applications" [RFC3413] for further information.

Throughout this document, the terms "client" and "server" are used to refer to the two ends of the TLS transport connection. The client actively opens the TLS connection, and the server passively listens for the incoming TLS connection. Either SNMP entity may act as

client or as server, as discussed further below.

According to the terminology used in RFC3411 [RFC3411], a principal is the "who" on whose behalf services are provided or processing takes place. A principal can be, among other things, an individual acting in a particular role; a set of individuals, with each acting in a particular role; an application or a set of applications, or a combination of these within an administrative domain.

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].

1.3. Modularity

The reader is expected to have read and understood the description of the SNMP architecture, as defined in [RFC3411], and the Transport Subsystem architecture extension specified in "Transport Subsystem for the Simple Network Management Protocol" [I-D.ietf-isms-tmsm].

This memo describes the Transport Layer Security Transport Model for SNMP, a specific SNMP transport model to be used within the SNMP transport subsystem to provide authentication, encryption, and integrity checking of SNMP messages.

In keeping with the RFC 3411 design decisions to use self-contained documents, this memo includes the elements of procedure plus associated MIB objects which are needed for processing the Transport Layer Security Transport Model for SNMP. These MIB objects SHOULD NOT be referenced in other documents. This allows the Transport Layer Security Transport Model for SNMP to be designed and documented as independent and self- contained, having no direct impact on other modules, and allowing this module to be upgraded and supplemented as the need arises, and to move along the standards track on different time-lines from other modules.

This modularity of specification is not meant to be interpreted as imposing any specific requirements on implementation.

1.4. Motivation

Version 3 of the Simple Network Management Protocol (SNMPv3) added security to the protocol. The User Security Model (USM) [RFC3414] was designed to be independent of other existing security infrastructures, to ensure it could function when third party authentication services were not available, such as in a broken network. As a result, USM typically utilizes a separate user and key management infrastructure. Operators have reported that deploying

another user and key management infrastructure in order to use SNMPv3 is a reason for not deploying SNMPv3 at this point in time.

This memo describes a transport model that will make use of the existing and commonly deployed X.509 key management infrastructure [RFC3280] to be used with the TLS security protocol [RFC4346]. This transport model is designed to meet the security and operational needs of network administrators, maximize usability in operational environments to achieve high deployment success and at the same time minimize implementation and deployment costs to minimize the time until deployment is possible.

This work will address the requirement of a TLS server to authenticate to a TLS client and a TLS client to authenticate to a TLS server. The TLS Record protocol [RFC4346], which runs on top of a reliable transport protocol is used to provide secure communication between a TLS client and a TLS server and to encapsulate higher level protocols. The TLS Handshake protocol, which runs on top of the TLS Record protocol, is used for authentication. The authentication is established by using X.509 [RFC3280] certificates that are exchanged between the TLS client and the TLS server.

The memo also specifies how X.509 certificates [RFC3280] are used in order to extract transport model specific security parameters, and how they are translated into transport model independent security parameters. Furthermode, a notion of a session is defined and specification how sessions are created, maintained, and possibly resumed is given.

There are a number of challenges to be addressed to map TLS authentication method parameters into the SNMP architecture so that SNMP continues to work without any surprises. These are discussed in detail below.

1.5. Constraints

The design of this SNMP Transport Model is influenced by the following constraints:

1. When the requirements of effective management in times of network stress are inconsistent with those of security, the design of this model gives preference to effective management.
2. In times of network stress, the transport protocol and its underlying security mechanisms SHOULD NOT depend upon the ready availability of other network services (e.g., Network Time Protocol (NTP) or AAA protocols).
3. When the network is not under stress, the transport model and its underlying security mechanisms MAY depend upon the ready availability of other network services.

4. It may not be possible for the transport model to determine when the network is under stress.
5. A transport model should require no changes to the SNMP architecture.
6. A transport model should require no changes to the underlying protocol.

2. The TLS Protocol

The TLS protocol [RFC4346] provides communications privacy over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. The primary goal of the TLS Protocol is to provide privacy and data integrity between two communicating applications. The protocol is composed of two layers: the TLS Record Protocol and the TLS Handshake Protocol.

The TLS Record Protocol runs on top of some reliable transport protocol (e.g., TCP). It provides connection security that has two basic properties: data privacy and message integrity. The TLS Record Protocol is used for encapsulation of various higher level protocols.

The TLS Handshake runs on top of the TLS Record Protocol. It allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data.

3. The X.509 Certificates Infrastructure

Users of a public key shall be confident that the associated private key is owned by the correct remote subject (person or system) with which an encryption or digital signature mechanism will be used. This confidence is obtained through the use of public key certificates, which are data structures that bind public key values to subjects. The binding is asserted by having a trusted CA digitally sign each certificate. The CA may base this assertion upon technical means (a.k.a., proof of posession through a challenge-response protocol), presentation of the private key, or on an assertion by the subject. A certificate has a limited valid lifetime which is indicated in its signed contents. Because a certificate's signature and timeliness can be independently checked by a certificate-using client, certificates can be distributed via untrusted communications and server systems, and can be cached in unsecured storage in certificate-using systems.

ITU-T X.509 (formerly CCITT X.509) or ISO/IEC/ITU 9594-8, which was

first published in 1988 as part of the X.500 Directory recommendations, defines a standard certificate format [X.509] which is a certificate which binds a subject (principal) to a public key value. A X.509 certificate is a sequence of three required fields:

1. tbsCertificate: The field contains the names of the subject and issuer, a public key associated with the subject, a validity period, and other associated information. This field may also contain extensions
2. signatureAlgorithm: The signatureAlgorithm field contains the identifier for the cryptographic algorithm used by the certificate authority (CA) to sign this certificate.
3. signatureValue: The signatureValue field contains a digital signature computed upon the ASN.1 DER encoded tbsCertificate field. The ASN.1 DER encoded tbsCertificate is used as the input to the signature function. This signature value is then ASN.1 encoded as a BIT STRING and included in the Certificate's signature field. By generating this signature, a CA certifies the validity of the information in the tbsCertificate field. In particular, the CA certifies the binding between the public key material and the subject of the certificate.

The basic X.509 authentication procedure is as follows: A system, which uses the X.509 key management infrastructure, is initialized with a number of root certificates which contain the public keys of a number of trusted CAs. When a system receives a X.509 certificate, signed by one of those CAs, that has to be verified, it first decrypts the signatureValue field by using the public key of the corresponding trusted CA. Then it compares the decrypted information with the tbsCertificate field. If they match then the subject in the tbsCertificate field is authenticated.

4. How TLSTM Fits into the Transport Subsystem

A transport model plugs into the Transport Subsystem. The TLS Transport Model will establish an encrypted tunnel between itself and the TLS Transport Model of another SNMP engine. The sending transport model passes unencrypted messages from the dispatcher to the TLS to be encrypted and the receiving transport model accepts decrypted messages from TLS and passes them to the dispatcher.

After an TLS Transport model tunnel is established, then SNMP messages can conceptually be sent through the tunnel from one SNMP message dispatcher to another SNMP message dispatcher. Multiple SNMP messages MAY be passed through the same tunnel.

The TLS Transport Model of an SNMP engine will perform the translation between TLS-specific security parameters and SNMP-

specific, model-independent parameters.

4.1. Security Capabilities of this Model

4.1.1. Threats

The TLS Transport Model provides protection against the threats identified by the RFC 3411 architecture [RFC3411]:

1. Information Modification - The TLS Record Layer [RFC4346] utilizes symmetric cryptography for data encryption. Message integrity check is provided by a keyed MAC which is computed before encryption by using secure hash functions (e.g.,SHA, MD5, etc.). Thus, TLS provides for verification that each received message has not been modified during its transmission through the network.
2. Message Stream Modification - The TLS Record Layer computes MAC of each record which also includes the sequence numbers. Thus, missing, extra or repeated messages are detectable.
3. Masquarade - The peer's identity can be authenticated using asymmetric, or public key, cryptography (e.g., RSA, DSS, etc.) by the TLS Handshake Protocol. For this purpose the X.509 certificates [RFC3280] can be used which bind a principal to a public key value. TLS allows for verification of the identity of both client and server authentication.
4. Disclosure - The negotiation of a shared secret performed by the TLS Handshake Protocol is secure: the negotiated secret is unavailable to eavesdroppers, and for any authenticated connection the secret cannot be obtained, even by an attacker who can place himself in the middle of the connection. The negotiation is reliable: no attacker can modify the negotiation communication without being detected by the parties to the communication.

4.1.2. Data Origin Authentication Issues

The RFC 3411 architecture recognizes three levels of security:

1. without authentication and without privacy (noAuthNoPriv)
2. with authentication but without privacy (authNoPriv)
3. with authentication and with privacy (authPriv)

The TLS protocol provides support for encryption and data integrity. While it is technically possible to support no authentication and no encryption in TLS it is NOT RECOMMENDED by [RFC4346].

The TLS Transport Model determines from TLS the identity of the authenticated principal, and the type and address associated with an incoming message, and the TLS Transport Model provides this information to TLS for an outgoing message. The transport layer

algorithms used to provide authentication, data integrity and encryption SHOULD NOT be exposed to the TLS Transport Model layer. The SNMPv3 WG deliberately avoided this and settled for an assertion by the security model that the requirements of securityLevel were met The TLS Transport Model has no mechanisms by which it can test whether an underlying TLS connection provides auth or priv, so the TLS Transport Model trusts that the underlying TLS connection has been properly configured to support authPriv security characteristics.

The TLS Transport Model does not know about the algorithms or options to open TLS sessions that match different securityLevels. For interoperability of the trust assumptions between SNMP engines, an TLS Transport Model-compliant implementation MUST use a TLS connection that provides authentication, data integrity and encryption that meets the highest level of SNMP security (authPriv). Outgoing messages requested by SNMP applications and specified with a lesser securityLevel (noAuthNoPriv or authNoPriv) are sent by the TLS Transport Model as authPriv securityLevel.

The security protocols used in the TLS Handshake Protocol [RFC4346] are considered acceptably secure at the time of writing. However, the procedures allow for new authentication and privacy methods to be specified at a future time if the need arises.

4.1.3. Authentication Protocol

The TLS Transport Model should support public key authentication via X.509 certificates. The user identity could be considered to be the Common Name field in the X.509 v3 certificate or any of the fields in the AlternativeName extension defined in [RFC3280]. The general algorithm described in [RFC3280] for X.509 certificate validation can be used.

4.1.4. Privacy Protocol

TLS uses symmetric cryptography for data encryption (e.g., DES, RC4, etc.) The keys for this symmetric encryption are generated uniquely for each connection and are based on a secret negotiated by another protocol (such as the TLS Handshake Protocol). The Record Protocol can also be used without encryption.

4.1.5. Protection against Message Replay, Delay and Redirection

The TLS Record Layer computes a MAC of each record which also includes the sequence numbers. Thus, missing, extra or repeated messages are detectable.

4.2. Security Parameter Passing

For incoming messages, TLS-specific security parameters are translated by the transport model into security parameters independent of the transport and security models. The transport model accepts messages from the TLS subsystem, and records the transport-related and TLS-security-related information, including the authenticated identity, in a cache referenced by tmStateReference, and passes the WholeMsg and the tmStateReference to the dispatcher using the recvMessage() ASI.

For outgoing messages, the transport model takes input provided by the dispatcher in the sendMessage() ASI. The TLS Transport Model converts that information into suitable security parameters for TLS, establishes sessions as needed, and passes messages to the TLS subsystem for sending.

4.3. Notifications and Proxy

TLS connections may be initiated by command generators or by notification originators. Command generators are frequently operated by a human, but notification originators are usually unmanned automated processes. As a result, it may be necessary to provision notification originators with X.509 certificates so that they can successfully authenticate to notification receivers by using TLS.

The targets to whom notifications should be sent is typically determined and configured by a network administrator. The SNMP-TARGET-MIB module [RFC3413] contains objects for defining management targets, including transport domains and addresses and security parameters, for applications such as notifications and proxy.

For the TLS Transport Model, transport domain and address are configured in the snmpTargetAddrTable, and the securityModel, securityName, and securityLevel parameters are configured in the snmpTargetParamsTable. The default approach is for an administrator to statically preconfigure this information to identify the targets authorized to receive notifications or perform proxy.

These MIB modules may be configured using SNMP or other implementation-dependent mechanisms, such as CLI scripting or loading a configuration file.

4.4. Provisioning for the X.509 PKI framework

Authentication using TLS will require that SNMP agents are provisioned with X.509 v3 certificates, which are signed by trusted certificate authorities. Furthermore, SNMP agents will have to be

provisioned with root certificates which represent the list of trusted certificate authorities that an SNMP agent can use for certificate verification. SNMP agents must also be provisioned with X.509 certificate revocation server which will be used to verify that a certificate has not been revoked.

5. Passing Security Parameters

For the TLS Transport Model, there are two levels of state that need to be maintained: the session state, and the message state.

5.1. tmStateReference

For each connection, the TLS Transport Model stores information about the connection in the Local Configuration Datastore (LCD), supplemented with a cache to store model- and mechanism-specific parameters.

Upon opening an TLS connection, the TLS Transport Model will store the transport parameters in the LCD.

tmsLCDTransport = the OID identifying the TLS transport domain

tmsLCDAddress = the address of the TLS transport endpoint

tmsLCDSecurityLevel = "authPriv"

tmsLCDSecurityName = the principal name authenticated by TLS. How this data is extracted from the TLS environment and how it is translated into a securityName is implementation-dependent. By default, the tmSecurityName is the subject that has been successfully authenticated by TLS from the Common Name field of the tbsCertificate field of the X.509 v3 certificate

tmsLCDEngineID = if known, the value of the remote engine's snmpEngineID.

tmsLCDSecurityModel = a security model. The TLS Transport Model is designed to work with multiple security models. The default is the Transport Security Model [I-D.ietf-isms-transport-security-model-02].

How the TLS identity is extracted from the TLS layer, and how the TLS identity is mapped to a securityName for storage in the LCD is implementation-dependent. Additional information may be stored in a local datastore (such as a preconfigured mapping table) or in a cache. Transport dependent security parameters may be stored in a cache or in the LCD such as a TLS session identifier which should be

extracted from the Client Hello and Server Hello messages during session establishment. TLS Security parameters that are used by the TLS Handshake protocol should also be stored in the LCD or in a cache. The TLS security parameters are defined in [RFC4346]:

```
struct {
      ConnectionEnd          entity;
      BulkCipherAlgorithm    bulk_cipher_algorithm;
      CipherType             cipher_type;
      uint8                  key_size;
      uint8                  key_material_length;
      MACAlgorithm           mac_algorithm;
      uint8                  hash_size;
      CompressionMethod      compression_algorithm;
      opaque                 master_secret[48];
      opaque                 client_random[32];
      opaque                 server_random[32];
  } SecurityParameters;
```

[discuss: Are those the right security parameters to cache for a session? These are provided by the TLS Handshake protocol to the TLS Record protocol so that the latter can generate keys and secrets as defined in RFC4346. However, can those parameters be extracted from the TLS layer? Are the above described parameters actually the negotiated parameters during the TLS Handshake on session establishment? According to RFC4346 parameters are negotiated by the TLS Handshake protocol on session establishment and after change_cipher_spec message is received those are communicated to the TLS Record Layer. However RFC4346 is not very specific what exactly is communicated from the TLS Handshake Layer to the TLS Record Layer as I assume that exactly the information that is communicated after after change_cipher_spec is what we need to cache. This security information will be used later so that application data is encrypted.

As a matter of fact I discovered that the TLS Handshake protocol negotiates the following parameters:

1. session id
2. peer certificate
3. compression method
4. cipher suite
5. master secret
6. is resumable

Those are specified at the beginning of section 7 in RFC4346 and I guess it might be better to use those as TLS specific parameters to cache as those are used to generate the above listed Security Parameters.]

The TLS Security Parameters and the TLS session identifier can be used for session resumption.

The tmStateReference is used to pass references containing the appropriate TLS session information from the transport model for subsequent processing.

The TLS Transport Model has the responsibility for explicitly releasing the complete tmStateReference and deleting the associated information from the LCD when the session is destroyed. However, the TLS Transport Model may not explicitly delete the information associated with a session when the session is closed because this information may be later used for session resumption.

The tmStateReference should also contain a flag which denotes whether a session is active or not. When the session is established the flag should be set and when the session is closed the flag should be reset. Since the LCD may store session information even after a session is closed (for the purpose of session resumption) this flag will tell the TLS transport model whether the information in the LCD belongs to an active session (i.e if there is a message which belongs to that session the TLS transport model should invoke the send message primitive) or the information belongs to a closed session, which can be resumed (in which case the TLS transport has to call the resumeSession() primitive before sending the message).

5.2. securityStateReference

For each message received, the TLS Transport Model caches message-specific TLS security information such that a Response message can be generated using the same security information, even if the Local Configuration Datastore is altered between the time of the incoming request and the outgoing response. The securityStateReference is used to preserve the data needed to generate a Response message with the same security information. This information includes the model-independent parameters (securityName, securityLevel, securityModel, transport address, transport domain, and engineID). The Message Processing Model has the responsibility for explicitly releasing the securityStateReference when such data is no longer needed. The securityStateReference cached data may be implicitly released via the generation of a response, or explicitly released by using the stateRelease primitive, as described in RFC 3411 section 4.5.1.

The TLS standard does not require that an TLS session be maintained nor that it be closed when the keys associated with the host or client associated with the session are changed. Some TLS implementations might close an existing session if the keys associated with the session change. For the TLS Transport Model, if

the session is closed between the time a Request is received and a Response message is being prepared, then the Response should be discarded.

6. Elements of Procedure

Abstract service interfaces have been defined by RFC 3411 to describe the conceptual data flows between the various subsystems within an SNMP entity. The TLS Transport Model uses some of these conceptual data flows when communicating between subsystems. These RFC 3411-defined data flows are referred to here as public interfaces.

To simplify the elements of procedure, the release of state information is not always explicitly specified. As a general rule, if state information is available when a message gets discarded, the message-state information should also be released. Due to the capability of TLS to restore sessions, session state may be cached after a session is closed. This information may be used to restore a session later on. A minimal set of session parameters to be stored in order to restore a session later on is:

1. transportAddress
2. transport domain
3. securityName
4. securityModel
5. securityLevel
6. EngineID

The TLS specific parameters are:

1. session ID
2. compression method
3. cipher suite
4. peer certificate
5. is resumable

It makes sense to cache the session state after a session is closed only if the "is resumable" field is on. This is negotiated during the Client Hello and Server Hello messages during TLS session establishment.

In the following Elements of Procedure, an error indication may return an OID and value for an incremented counter and a value for securityLevel, and values for contextEngineID and contextName for the counter, and the securityStateReference if the information is available at the point where the error is detected.

6.1. Procedures for an Incoming Message

For an incoming message, the TLS Transport Model will put information from the TLS layer into a Local Configuration Datastore referenced by tmStateReference.

1) The TLS Transport Model queries the associated TLS engine, in an implementation-dependent manner, to determine the transport and security parameters for the received message.

transportDomain = snmpTLSDomain
transportAddress = a TLS transport address
tmsTransportModel = TLS Transport Model
tmsSecurityLevel = "authPriv"
tmsSecurityName = the name of the principal authenticated by TLS. How the name is extracted from the TLS environment and how it is translated into a securityName is implementation-dependent. The tmSecurityName which has to be authenticated should be extracted from the Common Name field of an X.509 v3 certificate or from an AlternativeName extension of an X.509 v3 certificate.

2) If there is no entry in the LCD corresponding to the extracted (transportDomain, transportAddress, tmsTransportModel, tmsSecurityLevel, tmsSecurityName) then the TLS Transport Model creates an entry in the LCD and caches the TLS specific parameters for that session. A recommended set of TLS specific parameters to be cached is session ID, compression method, cipher suite, is resumable and peer certificate. The error conditions that might be encountered during the session establishment are specified in Section 6.3 where an exact procedure for session establishment is specified.

3) At that point the TLS Transport model performs translation between the transport model dependent security name i.e tmSecurityName and the transport model independent securityName. This translation is implementation specific.

The transport model then passes the message to the Dispatcher by using the following ASI of transport subsystem:

```
statusInformation =
receiveMessage(
IN   transportDomain       -- domain for the received message
IN   transportAddress      -- address for the received message
IN   wholeMessage          -- the whole SNMP message from TLS
IN   wholeMessageLength    -- the length of the SNMP message
IN   tmStateReference      -- reference to transport state
 )
```

6.2. Procedures for an Outgoing Message

The Dispatcher passes the information to the Transport Model using the ASI defined in the transport subsystem:

```
statusInformation =
sendMessage(
IN   destTransportDomain          -- transport domain to be used
IN   destTransportAddress         -- transport address to be used
IN   outgoingMessage              -- the message to send
IN   outgoingMessageLength        -- its length
IN   tmStateReference
)
```

The TLS Transport Model performs the following tasks:

1) Determine the target 5-tuple index by extracting the transportDomain, transportAddress, securityName, securityLevel, and securityModel from the tmStateReference.

2) Lookup the session in the Local Configuration Datastore using the target index.

3) If there is a session in the LCD and it is denoted as active then go to step 9) to send the message. If there is a session but it is denoted as not active then go to step 4) to resume the session. If there is no session in the LCD then one has to be created, go to step 6) to create a session.

4) If there is a session and the flag indicating whether the session is active is not set, then the information about that session belongs to an already closed session which can possibly be resumed. In that case resumeSession() should be called with the session parameters that have been retrieved from the LCD in step 1).

5) If resumeSession() is successful, go to step 9) to send the message. Otherwise, closeSession() should be called, all cached information that belongs to the session which has failed to be resumed should be cleared from the cache, and a fallback to openSession() in step 6) has to be performed.

6) If a session does not exist in the LCD or resumeSession() failed then a call to openSession() has to be performed.

7) If an error is returned from openSession(), then discard the message and return the error indication in the statusInformation. Increment an error counter. Message processing is terminated.

8) If openSession() is successful, then store any implementation-specific information in the LCD for subsequent use.

9) Pass the wholeMessage as application_data record to be encapsulated by the TLS Record Layer. Message processing is completed.

6.3. Establishing a Session

The Secure Shell Transport Model provides the following primitive to describe the data passed between the Transport Model and the TLS service. It is an implementation decision how such data is passed.

```
statusInformation =
openSession(
IN   destTransportDomain       -- transport domain to be used
IN   destTransportAddress      -- transport address to be used
IN   securityModel             -- Security Model to use
IN   securityName              -- on behalf of this principal
IN   securityLevel             -- Level of Security requested
IN   maxMessageSize            -- of the sending SNMP entity
OUT  tmStateReference
 )
```

The following describes the procedure to follow to establish a session between a client and server to run SNMP over TLS. This process is followed by any SNMP engine establishing a session for subsequent use.

This will be done automatically for an SNMP application that initiates a transaction, such as a Command Generator or a Notification Originator or a Proxy Forwarder.

The need to establish a session is never triggered by an application sending a response message, such as a Command Responder or Notification Receiver, because securityStateReference will always have the information for an existing session, identifiable via tmStateReference.

1) Using destTransportDomain and destTransportAddress, the client will initiate a TLS Handshake with the server. The client and the server will exchange Hello messages where the protocol version, the compression method, the cipher suite and the session ID will be negotiated. Random values will also be exchanged. As a new session is to be established the session ID sent by the client must be empty.

2) The client must authenticate the server. The TLS server provides an X.509 v3 certificate and the client verifies it by following the

procedure outlined in Section 3. If server authentication fails then an error indication is returned, and openSession() processing stops.

3) The server may send a key exchange message if the information in the X.509 certificate is not sufficient to negotiate a pre-master secret for the TLS connection.

4) The server will request from the client to authenticate by sending its X.509 v3 certificate after which it sends a ServerHelloDone TLS message.

5) The client must authenticate to the server by sending its X.509 v3 certificate. In case the client is pre-configured with multiple certificates to be used for different servers it must look up the authenticated server principal in the LCD. The authenticated server principal is extracted from the Common Name field of the X.509 v3 server certificate or the AlternativeName extenstion to the X.509 v3 server certificate. If client authentication fails then an error indication is returned, and openSession() processing stops.

6) The client must send a client key exchange message which is used to set the premaster secret for the TLS connection.

7) The client and the server must exchange ChangeCipherSpec messages. At this point both sides of the connection provide the security parameters negotiated by the TLS Handshake Protocol to the TLS Record Protocol. Immediately after this, the client and the server exchange Finished messages which are the first encrypted messages exchanged between the two sides. This denotes the end of the TLS Handshake Protocol and a TLS session is established.

8) Any TLS specific information about the newly established session should be retained in a cache so that it can be added to the LCD later. A recommended set of parameters is
1. destTransportDomain
2. destTransportAddress
3. session ID
4. compression method
5. cipher suite
6. peer certificate
7. is resumable

9) Create an entry in a Local Configuration Datastore containing the provided transportDomain, transportAddress, securityName, securityLevel, and securityModel, and TLS-speciifc parameters that have been saved in a cache in step 8 and create a tmStateReference to reference the entry. Set the flag that denotes that a session is currently active.

10) At this point an implementation MAY perform some type of engineID discovery to determine a mapping between the remote transport address, the TLS session, and a contextEngineID.

[discuss: Who actually does the check that the resumed/created session satisfies the requested security level?]

The contextEngineID of a remote engine needs to be "discovered" for use in request messages. USM, the mandatory-to-implement security model, can perform discovery of the snmpEngineIDs of adjacent engines using Reports (see [RFC3414] section 3.2 3b). Then the discovered snmpEngineID for the remote engine can be used as the contextEngineID in requests passed using the TLS Transport Model.

[discuss: should we point to the engineID discovery document?]

6.4. Session Resumption

The TLS Transport Model provides the following primitive to resume a previously established session.

```
 statusInformation =
   resumeSession(
   IN  tmStateReference         -- reference to a previous session
    )
```

The following describes the procedure to follow to resume a previously established session between a client and server to run SNMP over TLS. This process is followed by any SNMP engine resuming a session for subsequent use.

A session can be resumed by any of the sides that have participated in the session beforehand as far as the original session has been negotiated as "is resumable" during the initial Hello message exchange. A session can be resumed by an SNMP application which wishes to initiate a transaction such as Command Generator, Notification Originator or Proxy Forwarder. Furthermore, a session can also be resumed by an engine which has participated as a Command Responder or a Notification Receiver in a previous session.

[discuss: from RFC4346 I could not figure out if a session can be resumed by the server or at least it is not specified how this happens. Is it possible? If not then a Command Responder and Notification Receiver cannot resume a session.]

1) The client looks up transport specific security parameters in the LCD by using the tmStateReference and extracts session ID, destTransportDomain, destTransportAddress, compression method, cipher

suite, peer certificate and is resumable. If the "is resumable" option is not negotiated during the original session, resumeSession() is aborted and a fallback to openSession() is performed.

[discuss: do we want to specify whether the client and the server try to re-check the peer certificates i.e whether they have expired or whether they have not been revoked before agreeing on session resumption. This is not specified in the TLS session resumption and is a bit too much I guess]

2) The client uses destTransportDomain and destTransportAddress to initiate a TLS Handshake. In the Client Hello message the client provides the session ID, the cipher suite and the compression method which have been extracted from the LCD in step 1)

3) The server looks up in its cache for the session ID sent by the client and verifies the compression algorithm and the cipher suite. If the session ID does not match, then the server generates a new session ID and a fallback to openSession() is performed i.e a full handshake takes place between the client and the server.

4) If successful the server sends a ServerHello message which is followed by ChangeCipherSpec message exchange between the client and the server. During this phase the Handshake protocols of the client and the server load the cached security parameters into the TLS Record Layer. Thus it is important that ALL neccessary security parameters are cached if a session is to be resumed.

5) The client and the server exchange Finished messages which denotes the end of the TLS Handshake and the resumption of the TLS session. The client and the server must set the flag which denotes whether a session, which is stored in the LCD, is active.

[discuss: Who actually does the check that the resumed/created session satisfies the requested security level?]

6) At this point an implementation MAY perform some type of engineID discovery to determine a mapping between the remote transport address, the TLS session, and a contextEngineID. The engineID may be retrieved from a cache if it has been stored during the original session

6.5. Closing a Session

The TLS Transport Model provides the following primitive to close a session.

```
statusInformation =
closeSession(
IN  tmStateReference
 )
```

The following describes the procedure.

1) Determine the target 5-tuple index by extracting the transportDomain, transportAddress, securityName, securityLevel, and securityModel from the tmStateReference.

2) Lookup the session in the Local Configuration Datastore using the target index.

3) If there is no session open associated with the target index, then closeSession processing is completed.

4) Extract any implementation-specific parameters from the LCD. Among the extracted transport specific parameters should be the "is resumable" option which has been negotiated between the client and the server during the Hello Exchange messages in the original session. If the "is resumable" option is set, then the session parameters can be stored for later use if session resumption occurs. Reset the flag in the tmStateReference which indicates whether a session that is stored in the LCD is currently active.

5) Have the TLS close the connection. A client and a server must make a proper use of closure alerts by following the specification in section 7.2.1 from [RFC4346] during session teardown. The side which initiates the close must send a close_notify message. When the other side receives the close_notify message it must also respond with close_notify and discard any pending data after which the connection must be immediately closed. If a session is not closed by proper handling of closure alerts it cannot be resumed later on.

7. IANA Considerations

8. Security Considerations

In order to allow SNMP traffic to be easily identified and filtered by firewalls and other network devices, servers associated with SNMP entities using the TLS Transport Model MUST default to providing access to the "SNMP" TLS subsystem if the TLS session is established using the IANA-assigned TCP port (TBD by IANA). Servers SHOULD be configurable to allow access to the SNMP TLS subsystem over other ports.

9. Acknowledgements

This document closely follows the Secure Shell Transport Model for SNMP defined by David Harrington and Joseph Salowey in [I-D.ietf-isms-secshell]. Much the text was copied literally and then adapted to the specific needs of the TLS transport model.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, March 1997.

[RFC2578] McCloghrie, K., Perkins, D., and J. Schoenwaelder, "Structure of Management Information Version 2 (SMIv2)", RFC 2578, STD 58, April 1999.

[RFC2579] McCloghrie, K., Perkins, D., and J. Schoenwaelder, "Textual Conventions for SMIv2", RFC 2579, STD 58, April 1999.

[RFC2580] McCloghrie, K., Perkins, D., and J. Schoenwaelder, "Conformance Statements for SMIv2", RFC 2580, STD 58, April 1999.

[RFC3411] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

[RFC3280] Housley, R., Ford, W., Polk, T., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[I-D.ietf-isms-tmsm] Harrington, D. and J. Schoenwaelder, "Transport Subsystem for the Simple Network Management Protocol (SNMP)", February 2006.

[I-D.ietf-isms-transport-security-model-02] Harrington, D., "Transport Security Model for SNMP", January 2007.

10.2. Informative References

[RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, December 2002.

[RFC3413] Levi, D., Meyer, P., and B. Stewart, "Simple Network Management Protocol (SNMP) Applications", STD 62, RFC 3413, December 2002.

[RFC3414] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002.

[I-D.ietf-isms-secshell] Harrington, D. and J. Salowey, "Secure Shell Transport Model for SNMP", October 2006.

Authors' Addresses

Vladislav Marinov
Jacobs University Bremen
Campus Ring 1
28725 Bremen
Germany

Phone: +49 176 70046718
Email: v.marinov@iu-bremen.de

Juergen Schoenwaelder
Jacobs University Bremen
Campus Ring 1
28725 Bremen
Germany

Phone: +49 421 200-3587
Email: j.schoenwaelder@iu-bremen.de

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).