Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 14, 2007

D. Harrington
Huawei Technologies (USA)

October 11, 2006

Transport Security Model for SNMP
draft-ietf-isms-transport-security-model-00.txt

Status of This Memo

Copyright Notice

Abstract

This memo describes a Transport Security Model for the Simple Network Management Protocol.

Table of Contents

1. Introduction

This memo describes a Transport Security Model for the Simple Network Management Protocol, for use with secure transport models in the Transport Subsystem [I-D.ietf-isms-tmsm].

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 Transport Security Model for SNMP.

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

While security protocols frequently refer to a user, the terminology used in RFC3411 [RFC3411] and in this memo is "principal". 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].

Sections requiring further editing are identified by [todo] markers in the text. Points requiring further WG research and discussion are identified by [discuss] markers in the text.

1.3. Modularity

The reader is expected to have read and understood the description of the SNMP architecture, as defined in [RFC3411],and the architecture extension specified in "Transport Subsystem for the Simple Network Management Protocol" [I-D.ietf-isms-tmsm], which enables the use of external "lower layer transport" protocols to provide message security, tied into the SNMP architecture through the transport subsystem. The Transport Security Model is designed to work with such lower-layer secure transport models.

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 Security Model for SNMP. These MIB objects SHOULD not be referenced in other documents. This allows the Transport Security Model 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 previous versions of 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 security model that will make use of transport models that rely on lower layer secure transports and existing and commonly deployed security infrastructures. This security 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.

1.5. Constraints

The design of this SNMP Security Model is also 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 security 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 security model and its underlying security mechanisms MAY depend upon the ready availability of other network services.
4. It may not be possible for the security model to determine when the network is under stress.
5. A security model should require no changes to the SNMP architecture.
6. A security model should require no changes to the underlying security protocol.

2. How Transport Security Model Fits in the Architecture

The Transport Security Model is designed to fit into the RFC3411 architecture as a security model in the security subsystem, and to utilize the services of a secure transport model.

Within an engine using secure transport model, outgoing SNMP messages are passed unencrypted from the message dispatcher to the transport model, and incoming messages are passed unencrypted from the transport model to the message dispatcher.

[todo] locate and eliminate discussion of "dispatcher" functionality.

The transport model of an SNMP engine will perform the translation between transport-specific security parameters and SNMP-specific, model-independent parameters. Some security parameters may also be translated within a security model, for compatibility with the ASIs between the RFC 3411 Security Subsystem and the Message Processing

Subsystem.

2.1. Security Capabilities of this Model

2.1.1. Threats

The Transport Security Model, when used with suitable secure transport models, provides protection against the threats identified by the RFC 3411 architecture [RFC3411].

Which threats are addressed depends on the transport model. The Transport Security Model does not address any threats itself, but delegates that responsibility to a secure transport model.

The Transport Security Model is called a security model to be compatible with the RFC3411 architecure. However, this security model provides no security itself, so it SHOULD always be used with a transport model that provides appropriate security.

2.1.2. Security Levels

The RFC 3411 architecture recognizes three levels of security:

- without authentication and without privacy (noAuthNoPriv)
- with authentication but without privacy (authNoPriv)
- with authentication and with privacy (authPriv)

The model-independent securityLevel parameter is used to request specific levels of security for outgoing messages, and to assert that specific levels of security were applied during the transport and processing of incoming messages.

The transport layer algorithms used to provide security SHOULD NOT be exposed to the Transport Security Model, s the Transport Security Model has no mechanisms by which it can test whether an assertion made by a transport model is accurate.

The Transport Security Model trusts that the underlying secure transport connection has been properly configured to support security characteristics at least as strong as requested in securityLevel.

2.2. No Sessions

The Transport Security Model will associate state regarding each message and each known remote engine with a single combination of transportType, transportAddress, securityName, securityModel, and securityLevel.

Some transport models will utilize sessions to maintain long-lived

state; others will use stateless transport. For reasons of module independence, the Transport Security Model will make no assumptions about there being a session of any kind. Each message may be totally independent of other messages. Any binding of multiples messages into a session is specific to the transport model. There may be circumstances where having an snmp-specific session provided by a security model is useful; such functionality is left to future security models.

2.3. Coexistence

In RFC3411, there are dependencies between the message model and the security models; the security model fills in portions of the message, and thus must know the message format. When considering coexistence, one must consider coexistence with other message formats and other security models.

RFC3584 describes how to transfer fields between SNMPv3 and SNMPv1/v2c messages. The Transport Security Model usage of the msgSecurityParameters fields in SNMPv3 messages will be covered below.

The coexistence of the Transport Security Model with the community-based secruity used by SNMPv1 and SNMPv2c can be described in a different document.

The Transport Security Model can coexist with the USM security model, the only other currently defined security model. This can occur in multiple ways.

- o USM is combined with a non-secure transport model (the normal way to use USM). The SNMPv3 messaging model would pass the message to the USM security model for processing based on the securityModel expressed in msgSecurityParameters.
- o USM is combined with a secure transport model, which would encapsulate the whole SNMPv3 message to provide secure transport, but the global parameters specify USM as the security model, so SNMPv3 would pass the wholeMessage to the SNMPv3 messaging model. The SNMPv3 messaging model would pass the message to the USM security model for processing based on the securityModel expressed in msgSecurityParameters.
- o The SNMPv3 message specifies the Transport Security Model, so SNMPv3 sends the message to this security model for processing. Should secure transport models specify the securityModel as being the Transport Security Model, as well as determining the securityname and securityLevel? This has the unfortunate property of binding specific transport models to a specific security model. [discuss: this must be done by the transport model, because only

the transport model knows the message was secured at the transport level; it makes this assertion by setting securityLevel for incoming mesages, so it is possible that the transport **subsystem** could handle the choice of securityModel, if all transport models must default to an "unknown" securityLevel unless it actually does provide a known security level. Then the transport subsystem could check the securityLevel coming from the transport model, and if securityLevel is not "unknown", it could specify the transportSecurityModel. But this is a side-effect approach that could be problematic for future transport models and security models, so I prefer to not go that direction. It seems simpler to have the transport model specify it should be used with the Transport Security Model. An alternative could be to add a table in the trasnport subsystem that provided a mapping between transport model and security model, but that smacks of over-engineering]

2.4. Security Parameter Passing

For incoming messages, the transport model accepts messages from the lower layer transport, and records the transport-related information and security-related information, including the authenticated identity, in a cache referenced by tmStateReference. Then the transport model passes the WholeMsg and the tmStateReference to the security subsystem.

For outgoing messages, Transport Security Model takes input provided by the SNMP application, converts that information into suitable transport and security parameters, and passes these in a cache referenced by tmStateReference to the transport subsystem.

The cache reference is an additional parameter in the ASIs between the transport model and the security model. Passing a model-independent cache reference as a parameter in an ASI is consistent with the securityStateReference cache already being passed around in the ASI.

2.5. Notifications and Proxy

The SNMP-TARGET-MIB module [RFC3413] contains objects for defining management targets, including transportType, transportAddress, securityName, securityModel, and securityLevel parameters, for applications such as notifications and proxy. For the Transport Security Model, transport type 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.

3. Message Formats

The syntax of an SNMP message using this Security Model adheres to the message format defined in the version-specific Message Processing Model document (for example [RFC3412]). At the time of this writing, there are three defined message formats - SNMPv1, SNMPv2c, and SNMPv3. SNMPv1 and SNMPv2c have been declared Historic, so this memo only deals with SNMPv3 messages.

The processing is compatible with the RFC 3412 primitives, generateRequestMsg() and processIncomingMsg(), that show the data flow between the Message Processor and the security model.

3.1. SNMPv3 Message Fields

The SNMPv3Message SEQUENCE is defined in [RFC3412] and [RFC3416].

```
   SNMPv3MessageSyntax DEFINITIONS IMPLICIT TAGS ::= BEGIN

          SNMPv3Message ::= SEQUENCE {
              -- identify the layout of the SNMPv3Message
              -- this element is in same position as in SNMPv1
              -- and SNMPv2c, allowing recognition
              -- the value 3 is used for snmpv3
              msgVersion INTEGER ( 0 .. 2147483647 ),
              -- administrative parameters
              msgGlobalData HeaderData,
              -- security model-specific parameters
              -- format defined by Security Model
              msgSecurityParameters OCTET STRING,
              msgData  ScopedPduData
          }

          HeaderData ::= SEQUENCE {
              msgID      INTEGER (0..2147483647),
              msgMaxSize INTEGER (484..2147483647),

              msgFlags   OCTET STRING (SIZE(1)),
                         --  .... ...1   authFlag
                         --  .... ..1.   privFlag
                         --  .... .1..   reportableFlag
                         --              Please observe:
                         --  .... ..00   is OK, means noAuthNoPriv
                         --  .... ..01   is OK, means authNoPriv
                         --  .... ..10   reserved, MUST NOT be used.
                         --  .... ..11   is OK, means authPriv

              msgSecurityModel INTEGER (1..2147483647)
          }

          ScopedPduData ::= CHOICE {
              plaintext    ScopedPDU,
              encryptedPDU OCTET STRING  -- encrypted scopedPDU value
          }

          ScopedPDU ::= SEQUENCE {
              contextEngineID  OCTET STRING,
              contextName      OCTET STRING,
              data             ANY -- e.g., PDUs as defined in [RFC3416]
          }
      END
```

The following describes how Transport Security Model treats certain fields in the message:

3.1.1. msgGlobalData

The msgGlobalData values are set by the Message Processing model (e.g., SNMPv3 Message Processing), and are not modified by the Transport Security Model.

msgMaxSize is determined by the implementation.

For outgoing messages, msgSecurityModel is set by the Message Processing model (e.g., SNMPv3) to the IANA-assigned value for the Transport Security Model. See http://www.iana.org/assignments/snmp-number-spaces.

For outgoing messages, the value of msgFlags is set by the Message Processing model (e.g., SNMPv3 Message Processing), which is not necessarily the actual securityLevel applied to the message by the transport model.

For incoming messages, the value of msgFlags is determined by the Message Processing model (e.g., SNMPv3 Message Processing), and the value is passed in the securityLevel parameter in the ASI between the messaging model and the security model.

3.1.2. securityLevel and msgFlags

For an outgoing message, securityLevel is the requested security for the message, passed in the ASIs. If a Transport Model cannot provide the requested securityLevel, the model MUST describe a standard behavior that is followed for that situation. if the Transport Model is able to provide stronger than requested security, that may be acceptable. If the Transport Model cannot provide at least the requested level of security, the Transport Model MUST discard the request and SHOULD notify the message processing model that the request failed.

The msgFlags field in the SNMPv3 message is closely related to securityLevel. msgFlags is Messaging Model dependent, while securityLevel is Messaging Model independent. To maintain the separation between subsystems, the Transport Model SHOULD NOT modify Message Model dependent fields. As a result, msgFlags in the SNMPv3 message MAY reflect the requested securityLevel, not the actual securityLevel applied to the message by the Transport Model.

Part of the responsibility of a Security Model is to ensure that the actual security provided by the underlying transport layer security mechanisms is configured to meet or exceed the securityLevel requested. When the Security Model processes the incoming message, it should compare the securityLevel provided by the messaging model

to the securityLevel provided by the transport model in tmStateReference. If they differ, the Security Model should determine whether the securityLevel provided by the transport model is acceptable (e.g. the transport securityLevel is greater than or equal to the requested securityLevel. If not, it should discard the message. Depending on the model, the Security Model may issue a reportPDU with a model-specific counter.

3.1.3. msgSecurityParameters

Since message security is provided by a "lower layer", and the securityName parameter is always determined by the transport model from the lower layer authentication method, the SNMP message does not need to carry message security parameters within the msgSecurityParameters field.

The field msgSecurityParameters in SNMPv3 messages has a data type of OCTET STRING. To prevent its being used in a manner that could be damaging, such as for carrying a virus or worm, when used with Transport Security Model its value MUST be the BER serialization of a zero-length OCTET STRING.

```
   TransportSecurityParametersSyntax DEFINITIONS IMPLICIT TAGS
            ::= BEGIN

   TransportSecurityParameters ::=
          SEQUENCE {
                 OCTET STRING
          }
   END
```

3.2. Cached Information and References

The RFC3411 architecture uses caches to store dynamic model-specific information, and uses references in the ASIs to indicate in a model-independent manner which cached information must flow between subsystems.

There are two levels of state that may need to be maintained: the security state in a request-response pair, and potentially long-term state relating to transport and security.

This state is maintained in caches and a Local Configuration Datastore (LCD). 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 being processed gets discarded, the state related to that message should also be discarded, and if state information is available when a

relationship between engines is severed, such as the closing of a transport session, the state information for that relationship might also be discarded.

This document differentiates the tmStateReference from the securityStateReference. This document does not specify an implementation strategy, only an abstract discussion of the data that must flow between subsystems. An implementation MAY use one cache and one reference to serve both functions, but an implementer must be aware of the cache-release issues to prevent the cache from being released before a security or transport model has had an opportunity to extract the information it needs.

3.2.1. securityStateReference

From RFC3411: "For each message received, the Security Model caches the state 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.

A Message Processing Model has the responsibility for explicitly releasing the cached data if such data is no longer needed. To enable this, an abstract securityStateReference data element is passed from the Security Model to the Message Processing Model. The cached security data may be implicitly released via the generation of a response, or explicitly released by using the stateRelease primitive, as described in RFC3411 section 4.5.1."

The information saved should include the model-independent parameters (transportType, transportAddress, securityName, securityModel, and securityLevel), related security parameters, and other information needed to imatch the response with the request. 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."

If the transport model connection is closed between the time a Request is received and a Response message is being prepared, then the Response message MAY be discarded.

3.2.2. tmStateReference

For each message or transport session, information about the message security is stored in the Local Configuration Datastore (LCD), supplemented with a cache, to pass model- and mechanism-specific

parameters. The state referenced by tmStateReference may be saved across multiple messages, as compared to securityStateReference which is only saved for the life of a request-response pair of messages.

The format of the cache and the LCD are implementation-specific. For ease of explanation, this document defines a MIB module to conceptually represent the LCD, but this is not meant to contrain implementations from doing it differently.

It is expected that the LCD will allow lookup based on the combination of transportType, transportAddress, securityName, securityModel, and securityLevel. It is expected that the cache contain these values or contain pointers/references to entries in the LCD.

It is expected that a transport model may store transport-specific parameters in the LCD for subsequent usage.

4. 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.

4.1. Generating an Outgoing SNMP Message

This section describes the procedure followed by an RFC3411-compatible system whenever it generates a message containing a management operation (such as a request, a response, a notification, or a report) on behalf of a user.

```
statusInformation =          -- success or errorIndication
prepareOutgoingMessage(
IN  transportDomain          -- transport domain to be used
IN  transportAddress         -- transport address to be used
IN  messageProcessingModel   -- typically, SNMP version
IN  securityModel            -- Security Model to use
IN  securityName             -- on behalf of this principal
IN  securityLevel            -- Level of Security requested
IN  contextEngineID          -- data from/at this entity
IN  contextName              -- data from/in this context
IN  pduVersion               -- the version of the PDU
IN  PDU                      -- SNMP Protocol Data Unit
IN  expectResponse           -- TRUE or FALSE
IN  sendPduHandle            -- the handle for matching
                                incoming responses
OUT  destTransportDomain     -- destination transport domain
OUT  destTransportAddress    -- destination transport address
OUT  outgoingMessage         -- the message to send
OUT  outgoingMessageLength   -- its length
            )
```

The IN parameters of the prepareOutgoingMessage() ASI are used to pass information from the dispatcher (for the application subsystem) to the message processing subsystem.

The abstract service primitive from a Message Processing Model to a security model to generate the components of a Request message is generateRequestMsg(), as described in Section 4.2.

The abstract service primitive from a Message Processing Model to a Security Model to generate the components of a Response message is generateResponseMsg(), as described in Section 4.2.:

Upon completion of processing, the Transport Security Model returns statusInformation. If the process was successful, the completed message is returned, without any privacy and authentication applied yet. If the process was not successful, then an errorIndication is returned.

The OUT parameters are used to pass information from the message processing subsystem to the dispatcher and on to the transport subsystem:

4.2. Security Processing for an Outgoing Message

This section describes the procedure followed by the Transport Security Model.

The parameters needed for generating a message are supplied to the security model by the message processing model via the generateRequestMsg() or the generateResponseMsg() ASI. The Transport Subsystem architectural extension has added the transportDomain, transportAddress, and tmStateReference parameters to the original RFC3411 ASIs.

```
  statusInformation =                -- success or errorIndication
        generateRequestMsg(
        IN   messageProcessingModel  -- typically, SNMP version
        IN   globalData              -- message header, admin data
        IN   maxMessageSize          -- of the sending SNMP entity
        IN   transportDomain           -- as specified by application
        IN   transportAddress          -- as specified by application
        IN   securityModel           -- for the outgoing message
        IN   securityEngineID        -- authoritative SNMP entity
        IN   securityName            -- on behalf of this principal
        IN   securityLevel           -- Level of Security requested
        IN   scopedPDU               -- message (plaintext) payload
        OUT  securityParameters      -- filled in by Security Module
        OUT  wholeMsg                -- complete generated message
        OUT  wholeMsgLength          -- length of generated message
        OUT  tmStateReference      -- reference to session info
             )


statusInformation = -- success or errorIndication
        generateResponseMsg(
        IN   messageProcessingModel  -- typically, SNMP version
        IN   globalData              -- message header, admin data
        IN   maxMessageSize          -- of the sending SNMP entity
        IN   transportDomain           -- as specified by application
        IN   transportAddress          -- as specified by application
        IN   securityModel           -- for the outgoing message
        IN   securityEngineID        -- authoritative SNMP entity
        IN   securityName            -- on behalf of this principal
        IN   securityLevel           -- Level of Security requested
        IN   scopedPDU               -- message (plaintext) payload
        IN   securityStateReference  -- reference to security state
                                     -- information from original
                                     -- request
        OUT  securityParameters      -- filled in by Security Module
        OUT  wholeMsg                -- complete generated message
        OUT  wholeMsgLength          -- length of generated message
        OUT  tmStateReference        -- reference to session info
             )
```

- o statusInformation - An indication of whether the construction of the message was successful. If not it contains an indication of the problem.
- o messageProcessingModel - The SNMP version number for the message to be generated.
- o globalData - The message header (i.e., its administrative information). This data is opaque to Transport Security Model.
- o maxMessageSize - The maximum message size as included in the message. This data is not used by Transport Security Model.
- o transportDomain - as specified by the application.
- o transportAddress - as specified by the application.
- o securityEngineID - Transport Security Model always sets this to the snmpEngineID of the sending SNMP engine.
- o securityName - identifies a principal to be used for securing an outgoing message. The securityName has a format that is independent of the Security Model. In case of a response this parameter is ignored and the value from the securityStateReference cache is used.
- o securityLevel
- o scopedPDU - The message payload. The scopedPDU is opaque to Transport Security Model.
- o securityStateReference - A handle/reference to cachedSecurityData that is used when sending an outgoing Response message. This is the exact same securityStateReference as was generated by the Transport Security module when processing the incoming Request message to which this is the Response message.
- o securityParameters - Always set to empty by Transport Security Model.
- o wholeMsg - The fully encoded SNMP message ready for sending on the wire.
- o wholeMsgLength - The length of the encoded SNMP message (wholeMsg).
- o tmStateReference - a handle/reference to the session information to be passed to the transport model.

Note that the Transport Subystem architectural extension adds transportDomain, transportAddress, and tmStateReference to these ASIs.

1) verify that securityModel is transportSecurityModel. If not, then an error indication is returned to the calling message model, and security model processing stops for this message.
2) If there is a securityStateReference, then this is a response to a request, so extract the cached security data. This should include transportDomain, transportAddress, securityName, securityLevel, and securityModel, and a tmStateReference. At this point, the data cache referenced by the securityStateReference can be released.

[todo: is the securityStateReference still accessible? Doesn't the MPM release the cache before calling the security model?]
3) If there is no securityStateReference, then find or create an entry in a Local Configuration Datastore containing the provided transportDomain, transportAddress, securityName, securityLevel, and securityModel, and create a tmStateReference to reference the entry.
4) fill in the securityParameters with the serialization of a zero-length OCTET STRING.
5) Combine the message parts into a wholeMsg and calculate wholeMsgLength.
6) The completed message (wholeMsg) with its length (wholeMsgLength) and securityParameters (a zero-length octet string) and tmStateReference is returned to the calling messaging model with the statusInformation set to success.

4.3. Processing an Incoming SNMP Message

4.4. Prepare Data Elements from Incoming Messages

The abstract service primitive from the Dispatcher to a Message Processing Model for a received message is:

```
result =                       -- SUCCESS or errorIndication
prepareDataElements(
IN   transportDomain           -- origin transport domain
IN   transportAddress          -- origin transport address
IN   wholeMsg                  -- as received from the network
IN   wholeMsgLength            -- as received from the network
IN   tmStateReference        -- from the transport model
OUT  messageProcessingModel    -- typically, SNMP version
OUT  securityModel             -- Security Model to use
OUT  securityName              -- on behalf of this principal
OUT  securityLevel             -- Level of Security requested
OUT  contextEngineID           -- data from/at this entity
OUT  contextName               -- data from/in this context
OUT  pduVersion                -- the version of the PDU
OUT  PDU                       -- SNMP Protocol Data Unit
OUT  pduType                   -- SNMP PDU type
OUT  sendPduHandle             -- handle for matched request
OUT  maxSizeResponseScopedPDU  -- maximum size sender can accept
OUT  statusInformation         -- success or errorIndication
                               -- error counter OID/value if error
OUT  stateReference            -- reference to state information
                               -- to be used for possible Response
)
```

Note that tmStateReference has been added to this ASI.

4.5. Security Processing for an Incoming Message

This section describes the procedure followed by the Transport Security Model whenever it receives an incoming message containing a management operation on behalf of a user from a Message Processing model.

The Message Processing Model extracts some information from the wholeMsg. The abstract service primitive from a Message Processing Model to the Security Subsystem for a received message is::

```
statusInformation =  -- errorIndication or success
                         -- error counter OID/value if error
processIncomingMsg(
IN   messageProcessingModel    -- typically, SNMP version
IN   maxMessageSize            -- of the sending SNMP entity
IN   securityParameters        -- for the received message
IN   securityModel             -- for the received message
IN   securityLevel             -- Level of Security
IN   wholeMsg                  -- as received on the wire
IN   wholeMsgLength            -- length as received on the wire
IN   tmStateReference          -- from the transport model
OUT  securityEngineID          -- authoritative SNMP entity
OUT  securityName              -- identification of the principal
OUT  scopedPDU,                -- message (plaintext) payload
OUT  maxSizeResponseScopedPDU  -- maximum size sender can handle
OUT  securityStateReference    -- reference to security state
 )                          -- information, needed for response
```

1) If the received securityParameters is not the serialization of an OCTET STRING formatted according to the transportSecurityParameters, and the contained OCTET STRING is not empty, then the snmpInASNParseErrs counter [RFC3418] is incremented, and an error indication (parseError) is returned to the calling module.

2) [todo: discuss how to compare the requested security parameters (extracted from msgFlags by the MPM), and the transport-model-provided actual security (reported in tmStateReference); compare securityLevel, securityModel, and securityName. While a different user-name may be used during authnentication, the tmStateReference should contain the model-independent securityName. This does imply we need to provide the securityName in the securityParameters of the SNMPv3 message, right?

2) Extract the value of securityName from the Local Configuration Datastore entry referenced by tmStateReference.

1) The securityEngineID is set to the local snmpEngineID, to satisfy the SNMPv3 message processing model in RFC 3412 section 7.2 13a).

3) The scopedPDU component is extracted from the wholeMsg.

4) The maxSizeResponseScopedPDU is calculated. This is the maximum size allowed for a scopedPDU for a possible Response message.

5)The security data is cached as cachedSecurityData, so that a possible response to this message can and will use the same security parameters. Then securityStateReference is set for subsequent reference to this cached data. For Transport Security Model, the securityStateReference should include a reference to the tmStateReference.

4) The statusInformation is set to success and a return is made to the calling module passing back the OUT parameters as specified in the processIncomingMsg primitive.

5. Overview

This MIB module provides management of the Transport Security Model. It defines some needed textual conventions, and some statistics.

5.1. Structure of the MIB Module

Objects in this MIB module are arranged into subtrees. Each subtree is organized as a set of related objects. The overall structure and assignment of objects to their subtrees, and the intended purpose of each subtree, is shown below.

5.2. Textual Conventions

Generic and Common Textual Conventions used in this document can be found summarized at http://www.ops.ietf.org/mib-common-tcs.html

5.3. The transportStats Subtree

This subtree contains counters specific to the Transport Security Model.

This subtree provides information for identifying fault conditions and performance degradation.

5.4. The transportState Subtree

This subtree contains information specific to state related to tmStateReference. Most of the state referenced by tmStateReference

should be transport-model-specific, and not needed here.

5.5. Relationship to Other MIB Modules

Some management objects defined in other MIB modules are applicable to an entity implementing Transport Security Model. In particular, it is assumed that an entity implementing Transport Security Model will implement the SNMPv2-MIB [RFC3418], the SNMP-FRAMEWORK-MIB [RFC3411] and the Transport-Subsystem-MIB [I-D.ietf-isms-tmsm].

5.5.1. Relationship to the SNMPv2-MIB

The 'system' group in the SNMPv2-MIB [RFC3418] is defined as being mandatory for all systems, and the objects apply to the entity as a whole. The 'system' group provides identification of the management entity and certain other system-wide data. The TSM-MIB does not duplicate those objects.

5.5.2. Relationship to the SNMP-FRAMEWORK-MIB

[todo] if the TSM-MIB does not actually have dependencies on SNMP-FRAMEWORK-MIB other than imports, then remove this paragraph.

5.5.3. Relationship to the Transport-Subsystem-MIB

The 'tmsmSession' group in the Transport-Subsystem-MIB [I-D.ietf-isms-tmsm] is defined as being applicable to all Transport Models. [todo] if the MIB module defined here does not actually have dependencies on Transport-Subsystem-MIB other than imports, then remove this paragraph.

5.5.4. MIB Modules Required for IMPORTS

The following MIB module imports items from [RFC2578], [RFC2579], [RFC2580], [RFC3411], [RFC3419], and [I-D.ietf-isms-tmsm]

6. MIB module definition

```
TSM-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    OBJECT-IDENTITY, mib-2, Counter32, Integer32
      FROM SNMPv2-SMI
    TestAndIncr, AutonomousType
      FROM SNMPv2-TC
    MODULE-COMPLIANCE, OBJECT-GROUP
```

```
      FROM SNMPv2-CONF
    SnmpAdminString,  SnmpSecurityLevel, SnmpEngineID
       FROM SNMP-FRAMEWORK-MIB
    TransportAddress, TransportAddressType
      FROM TRANSPORT-ADDRESS-MIB
    TransportAddressSSH, transportDomainSSH
      FROM SSHTM-MIB
    ;

tsmMIB MODULE-IDENTITY
    LAST-UPDATED "200509020000Z"
    ORGANIZATION "ISMS Working Group"
    CONTACT-INFO "WG-EMail:   isms@lists.ietf.org
                  Subscribe:  isms-request@lists.ietf.org

               Chairs:
                 Juergen Quittek
                 NEC Europe Ltd.
                 Network Laboratories
                 Kurfuersten-Anlage 36
                 69115 Heidelberg
                 Germany
                 +49 6221 90511-15
                  quittek@netlab.nec.de

                  Juergen Schoenwaelder
                  International University Bremen
                  Campus Ring 1
                  28725 Bremen
                  Germany
                  +49 421 200-3587
                  j.schoenwaelder@iu-bremen.de

               Editor:
                  David Harrington
                  Effective Software
                  50 Harding Rd
                  Portsmouth, New Hampshire 03801
                  USA
                  +1 603-436-8634
                  ietfdbh@comcast.net
                    "
       DESCRIPTION  "The Secure Shell Security Model MIB

                     Copyright (C) The Internet Society (2006). This
                     version of this MIB module is part of RFC XXXX;
                     see the RFC itself for full legal notices.
-- NOTE to RFC editor: replace XXXX with actual RFC number
```

```
--                     for this document and remove this note
                    "

       REVISION     "200509020000Z"         -- 02 September 2005
       DESCRIPTION  "The initial version, published in RFC XXXX.
-- NOTE to RFC editor: replace XXXX with actual RFC number
--                     for this document and remove this note
                    "

    ::= { mib-2 xxxx }
-- RFC Ed.: replace xxxx with IANA-assigned number and
--          remove this note

-- ---------------------------------------------------------- --
-- subtrees in the TSM-MIB
-- ---------------------------------------------------------- --

tsmNotifications OBJECT IDENTIFIER ::= { tsmMIB 0 }
tsmMIBObjects       OBJECT IDENTIFIER ::= { tsmMIB 1 }
tsmConformance   OBJECT IDENTIFIER ::= { tsmMIB 2 }

-- -------------------------------------------------------------
-- Objects
-- -------------------------------------------------------------

-- Statistics for the Transport Security Model


tsmStats         OBJECT IDENTIFIER ::= { tsmMIBObjects 1 }

-- [todo] do we need any stats?

-- The tsmUser Group ************************************************

tsmUser          OBJECT IDENTIFIER ::= { tsmMIBObjects 2 }

tsmUserSpinLock  OBJECT-TYPE
    SYNTAX       TestAndIncr
    MAX-ACCESS   read-write
    STATUS       current
    DESCRIPTION "An advisory lock used to allow several cooperating
                 Command Generator Applications to coordinate their
                 use of facilities to alter the tsmUserTable.
                "
    ::= { tsmUser 1 }

-- The table of valid users for the SSH Transport Model ********
```

```
tsmUserTable     OBJECT-TYPE
    SYNTAX       SEQUENCE OF tsmUserEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "The table of users configured in the SNMP engine's
                 Local Configuration Datastore (LCD).

                 To create a new user (i.e., to instantiate a new
                 conceptual row in this table), it is recommended to
                 follow this procedure:

                   1)  GET(tsmUserSpinLock.0) and save in sValue.
                   2)  SET(tsmUserSpinLock.0=sValue,
                           tsmUserStatus=createAndWait)

                 Finally, activate the new user:

                   3) SET(tsmUserStatus=active)

                 The new user should now be available and ready to be
                 used for SNMPv3 communication.

                 The use of tsmUserSpinlock is to avoid conflicts with
                 another SNMP command generator application which may
                 also be acting on the tsmUserTable.
                "
    ::= { tsmUser 2 }

tsmUserEntry     OBJECT-TYPE
    SYNTAX       tsmUserEntry
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION "A user configured in the SNMP engine's Local
                 Configuration Datastore (LCD) for the Session
                 Security Model.
                "
    INDEX       { tsmUserSecurityName }
    ::= { tsmUserTable 1 }

tsmUserEntry ::= SEQUENCE
    {
        tsmUserSecurityName     SnmpAdminString,
        tsmUserStorageType      StorageType,
        tsmUserStatus           RowStatus
    }


tsmUserSecurityName OBJECT-TYPE
```

```
    SYNTAX       SnmpAdminString
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "A human readable string representing the user in
                 Security Model independent format.

                 [todo: Wehn used with transport models that perform
                 authentication, the tsmUserSecurityName is the
                 securityName passed in tmStateReference.
                "
    ::= { tsmUserEntry 1 }


tsmUserStorageType OBJECT-TYPE
    SYNTAX       StorageType
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "The storage type for this conceptual row.

                 It is an implementation issue to decide if a SET for
                 a readOnly or permanent row is accepted at all. In some
                 contexts this may make sense, in others it may not. If
                 a SET for a readOnly or permanent row is not accepted
                 at all, then a 'wrongValue' error must be returned.
                "
    DEFVAL      { nonVolatile }
    ::= { tsmUserEntry 4 }

tsmUserStatus    OBJECT-TYPE
    SYNTAX       RowStatus
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION "The status of this conceptual row.

                 Until instances of all corresponding columns are
                 appropriately configured, the value of the
                 corresponding instance of the tsmUserStatus column
                 is 'notReady'.

                 The RowStatus TC [RFC2579] requires that this
                 DESCRIPTION clause states under which circumstances
                 other objects in this row can be modified:

                 The value of this object has no effect on whether
                 other objects in this conceptual row can be modified.
                 "
    ::= { tsmUserEntry 5 }
```

```
-- ------------------------------------------------------------
-- tsmMIB - Conformance Information
-- ------------------------------------------------------------

tsmGroups OBJECT IDENTIFIER ::= { tsmConformance 1 }

tsmCompliances OBJECT IDENTIFIER ::= { tsmConformance 2 }

-- ------------------------------------------------------------
-- Units of conformance
-- ------------------------------------------------------------
tsmGroup OBJECT-GROUP
    OBJECTS {

    }
    STATUS      current
    DESCRIPTION "A collection of objects for maintaining
                 information of an SNMP engine which implements the
                 SNMP Transport Security Model.
                "

    ::= { tsmGroups 2 }

-- ------------------------------------------------------------
-- Compliance statements
-- ------------------------------------------------------------

tsmCompliance MODULE-COMPLIANCE
    STATUS      current
    DESCRIPTION
        "The compliance statement for SNMP engines that support the
        TSM-MIB"
    MODULE
        MANDATORY-GROUPS { tsmGroup }
    ::= { tsmCompliances 1 }

END
```

7. Security Considerations

This document describes a security model that permits SNMP to utilize security services provided through an SNMP transport model. The Transport Security Model relies on transport models for mutual authentication, binding of keys, confidentiality and integrity. The security threats and how those threats are mitigated should be covered in detail in the specification of the transport model and the underlying secure transport.

Transport Security Model relies on a transport model to provide an authenticated principal for mapping to securityName, and an assertion for mapping to securityLevel, for access control purposes.

The Transport Security Model is called a security model to be compatible with the RFC3411 architecure. However, this security model provides no security itself. It SHOULD always be used with a transport model that provides security, but this is a run-time decision of the operator or management application, or a configuration decision of an operator.

7.1. MIB module security

There are a number of management objects defined in this MIB module with a MAX-ACCESS clause of read-write and/or read-create. Such objects may be considered sensitive or vulnerable in some network environments. The support for SET operations in a non-secure environment without proper protection can have a negative effect on network operations. These are the tables and objects and their sensitivity/vulnerability:
o [todo]

There are no management objects defined in this MIB module that have a MAX-ACCESS clause of read-write and/or read-create. So, if this MIB module is implemented correctly, then there is no risk that an intruder can alter or create any management objects of this MIB module via direct SNMP SET operations.

Some of the readable objects in this MIB module (i.e., objects with a MAX-ACCESS other than not-accessible) may be considered sensitive or vulnerable in some network environments. It is thus important to control even GET and/or NOTIFY access to these objects and possibly to even encrypt the values of these objects when sending them over the network via SNMP. These are the tables and objects and their sensitivity/vulnerability:
o [todo]

SNMP versions prior to SNMPv3 did not include adequate security. Even if the network itself is secure (for example by using IPSec or SSH), even then, there is no control as to who on the secure network is allowed to access and GET/SET (read/change/create/delete) the objects in this MIB module.

It is RECOMMENDED that implementers consider the security features as provided by the SNMPv3 framework (see [RFC3410] section 8), including full support for the USM and Transport Security Model cryptographic mechanisms (for authentication and privacy).

Further, deployment of SNMP versions prior to SNMPv3 is NOT RECOMMENDED. Instead, it is RECOMMENDED to deploy SNMPv3 and to enable cryptographic security. It is then a customer/operator responsibility to ensure that the SNMP entity giving access to an instance of this MIB module is properly configured to give access to the objects only to those principals (users) that have legitimate rights to indeed GET or SET (change/create/delete) them.

8. IANA Considerations

IANA is requested to assign:

1. an SMI number under mib-2, for the MIB module in this document,
2. an SnmpSecurityModel for the Transport Security Model, as documented in the MIB module in this document,

9. Acknowledgements

10. References

10.1. Normative References

[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[RFC2578]	McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.
[RFC2579]	McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.
[RFC2580]	McCloghrie, K., Perkins, D., and J. Schoenwaelder, "Conformance Statements for SMIv2", STD 58, RFC 2580, 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.
[RFC3412]	Case, J., Harrington, D., Presuhn, R., and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3412, 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.
[RFC3416]	Presuhn, R., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, December 2002.
[RFC3418]	Presuhn, R., "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, December 2002.
[RFC3419]	Daniele, M. and J. Schoenwaelder, "Textual Conventions for Transport Addresses", RFC 3419, December 2002.
[I-D.ietf-isms-tmsm]	Harrington, D. and J. Schoenwaelder, "Transport Mapping Security Model (TMSM) Architectural Extension for the Simple Network Management Protocol (SNMP)", draft-ietf-isms-tmsm-03 (work in progress), June 2006.

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.

Appendix A. Notification Tables Configuration

The SNMP-TARGET-MIB and SNMP-NOTIFICATION-MIB [RFC3413] are used to configure notification originators with the destinations to which notifications should be sent.

Most of the configuration is security-model-independent and transport-model-independent.

The values we will use in the examples for the five model-independent security and transport parameters are:

```
   transportType = transportDomainSSH
```

```
      transportAddress = 10.0.0.1:162
      securityModel = Transport Security Model
      securityName = sampleUser
      securityLevel = authPriv
```

The following example will configure the Notification Originator to send informs to a Notification Receiver at host 10.0.0.1 port 162 using the securityName "sampleUser". The columns marked with a "*" are the items that are Security Model or Transport Model specific.

The configuration for the "sampleUser" settings in the SNMP-VIEW-BASED-ACM-MIB objects are not shown here for brevity. First we configure which type of notification should be sent for this taglist (toCRTag). In this example, we choose to send an Inform.

```
   (preamble)
     snmpNotifyTable row:
          snmpNotifyName                 CRNotif
          snmpNotifyTag                  toCRTag
          snmpNotifyType                 inform
          snmpNotifyStorageType          nonVolatile
          snmpNotifyColumnStatus         createAndGo
   (postamble)
```

Then we configure a transport address to which notifications associated with this taglist should be sent, and we specify which snmpTargetParamsEntry should be used (toCR) when sending to this transport address.

```
   (preamble)
          snmpTargetAddrTable row:
             snmpTargetAddrName              toCRAddr
         *   snmpTargetAddrTDomain           transportDomainSSH
             snmpTargetAddrTAddress          10.0.0.1:162
             snmpTargetAddrTimeout           1500
             snmpTargetAddrRetryCount        3
             snmpTargetAddrTagList           toCRTag
             snmpTargetAddrParams            toCR   (must match below)
             snmpTargetAddrStorageType       nonVolatile
             snmpTargetAddrColumnStatus      createAndGo


   (postamble)
```

Then we configure which prinicipal at the host should receive the notifications associated with this taglist. Here we choose "sampleUser", who uses the Transport Security Model.

```
   (preamble)
         snmpTargetParamsTable row:
             snmpTargetParamsName            toCR
             snmpTargetParamsMPModel         SNMPv3
         *   snmpTargetParamsSecurityModel   TransportSecurityModel
         *   snmpTargetParamsSecurityName    "sampleUser"
             snmpTargetParamsSecurityLevel   authPriv
             snmpTargetParamsStorageType     nonVolatile
             snmpTargetParamsRowStatus       createAndGo


   (postamble)
```

A.1. Security Model Configuration

In the Transport Security Model MIB module (TSM-MIB), we configure the Security Model Parameters. Since we are using a Transport Security Model, we provide a pointer to the appropriate transport model entry

[discuss: there are problems here. Users need additional qualification, such as address/user or engineID/user. This is transport-model specific. While we identify the securtyModel, there is no ppace that has the mapping from transport model to MIB module. A RowPointer would be more accurate, and able to support multiple multi-field indices, such as engineID/user or address/user, but it would be harder for an operator to configure. In addition, the transport model needs to be able to lookup an entry in the LCD, given the address it received a message from, and the username used to perform authentication. if that name is not the same as the securityName, then there needs to be a table to perform the username-to-securityName conversion, and another to perform securityname-to-username conversion.]

```
   (preamble)
   tsmUserEntry ::= SEQUENCE
       {
           tsmUserSecurityName     "sampleUser"
           tmsUserTransportModel   transportDomainSSH
           tsmUserTransportParams  "sshUser"
           tsmUserStorageType      StorageType,
           tsmUserStatus           RowStatus
       }
   An Entry from the Transport Security Model MIB module
```

A.2. Transport Model Configuration

In the Secure Shell Transport Model MIB module (SSH-TM-MIB), we configure the transport model parameters.

```
(preamble)
sshtmUserEntry ::= SEQUENCE
    {
        sshtmUserName             sshUser
        sshtmUserSecurityName     "sampleUser"
        sshtmUserStorageType      StorageType,
        sshtmUserStatus           RowStatus
    }
```

An entry from the SSH Transport Model MIB module

Appendix B. Change Log

From SSHSM-04- to Transport-security-model-00

added tsmUserTable
updated Appendix - Notification Tables Configuration
remove open/closed issue appendices
changed tmSessionReference to tmStateReference

Author's Address

David Harrington
Huawei Technologies (USA)
1700 Alma Dr. Suite 100
Plano, TX 75075
USA

Phone: +1 603 436 8634
EMail: dharrington@huawei.com

Acknowledgement

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