Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 16, 2007

D. Harrington
Huawei Technologies (USA)
J. Schoenwaelder
International University Bremen
December 13, 2006

Transport Subsystem for the Simple Network Management Protocol (SNMP)
draft-ietf-isms-tmsm-05

Status of This Memo

Copyright Notice

Abstract

This document describes a Transport Subsystem, extending the Simple Network Management Protocol (SNMP) architecture defined in RFC 3411. This document describes a subsystem to contain transport models, comparable to other subsystems in the RFC3411 architecture. As work is being done to expand the transport to include secure transport such as SSH and TLS, using a subsystem will enable consistent design and modularity of such transport models. This document identifies

and discusses some key aspects that need to be considered for any transport model for SNMP.

Table of Contents

1. Introduction

This document describes a Transport Subsystem, extending the Simple Network Management Protocol (SNMP) architecture defined in [RFC3411]. This document identifies and discusses some key aspects that need to be considered for any transport model for SNMP.

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].

1.2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Motivation

There are multiple ways to secure one's home or business, in a continuum of alternatives. Let's consider three general approaches. In the first approach, an individual could buy a gun, learn to use it, and sit on your front porch waiting for intruders. In the second approach, one could hire an employee with a gun, schedule the employee, position the employee to guard what you want protected, hire a second guard to cover if the first gets sick, and so on. In the third approach, you could hire a security company, tell them what you want protected, and they could hire employees, train them, buy the guns, position the guards, schedule the guards, send a replacement when a guard cannot make it, etc., thus providing the security you want, with no significant effort on your part other than identifying requirements and verifying the quality of the service being provided.

The User-based Security Model (USM) as defined in [RFC3414] largely uses the first approach - it provides its own security. It utilizes existing mechanisms (SHA=the gun), but provides all the coordination. USM provides for the authentication of a principal, message encryption, data integrity checking, timeliness checking, etc.

USM was designed to be independent of other existing security infrastructures. USM therefore requires a separate principal and key management infrastructure. Operators have reported that deploying another principal and key management infrastructure in order to use SNMPv3 is a deterrent to deploying SNMPv3. It is possible but difficult to define external mechanisms that handle the distribution

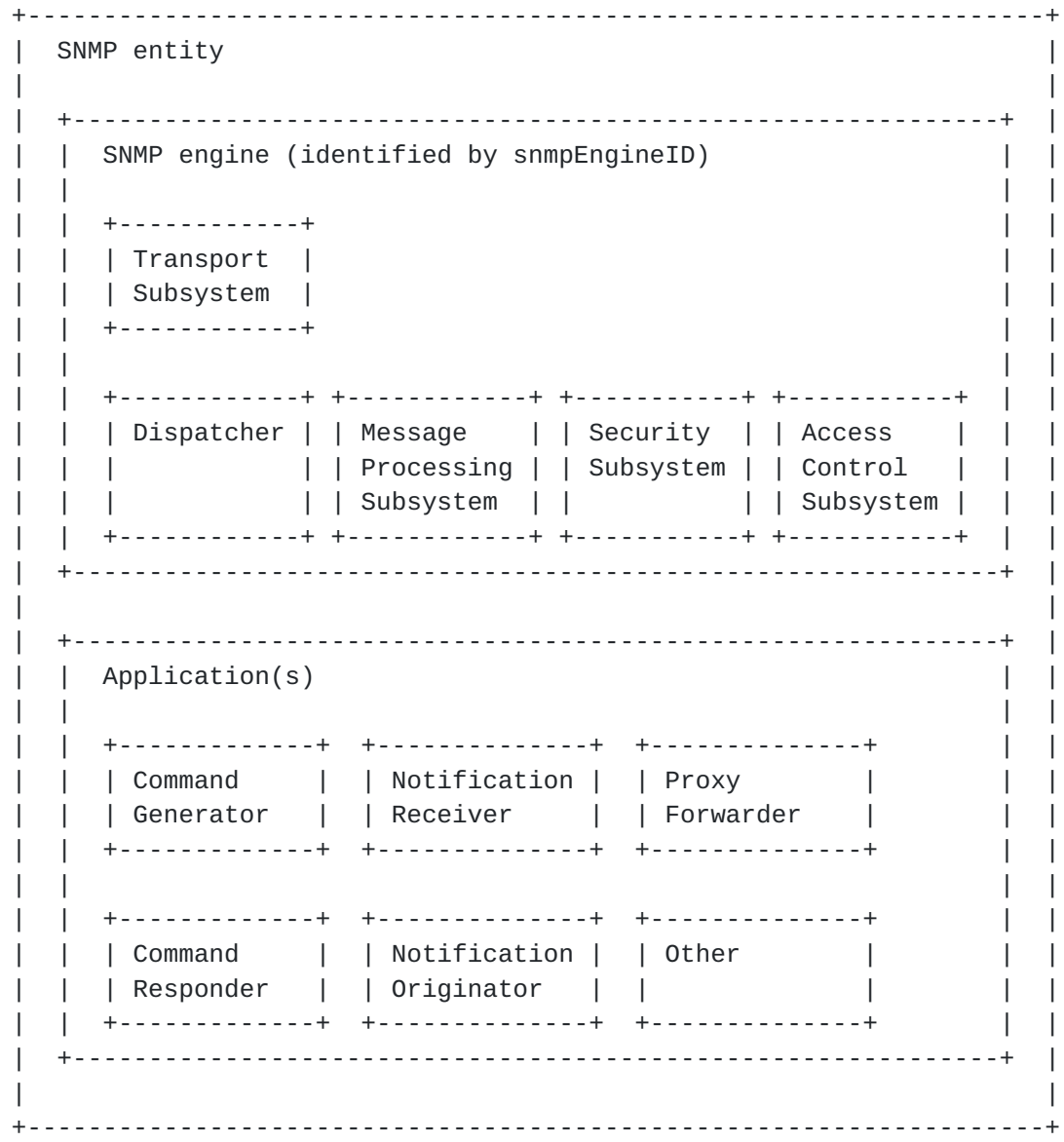
of keys for use by the USM approach.

A solution based on the second approach might use a USM-compliant architecture, but combine the authentication mechanism with an external mechanism, such as RADIUS [RFC2865], to provide the authentication service. It might be possible to utilize an external protocol to encrypt a message, to check timeliness, to check data integrity, etc. It is difficult to cobble together a number of subcontracted services and coordinate them however, because it is difficult to build solid security bindings between the various services, and potential for gaps in the security is significant.

A solution based on the third approach might utilize one or more lower-layer security mechanisms to provide the message-oriented security services required. These would include authentication of the sender, encryption, timeliness checking, and data integrity checking. There are a number of IETF standards available or in development to address these problems through security layers at the transport layer or application layer, among them TLS [RFC4366], SASL [RFC4422], and SSH [RFC4251].

From an operational perspective, it is highly desirable to use security mechanisms that can unify the administrative security management for SNMPv3, command line interfaces (CLIs) and other management interfaces. The use of security services provided by lower layers is the approach commonly used for the CLI, and is also the approach being proposed for NETCONF [I-D.ietf-netconf-ssh].

This document describes a Transport Subsystem extension to the RFC3411 architecture.



This extension allows security to be provided by an external protocol connected to the SNMP engine through an SNMP transport-model [RFC3417]. Such a transport model would then enable the use of existing security mechanisms such as (TLS) [RFC4366] or SSH [RFC4251] within the RFC3411 architecture.

There are a number of Internet security protocols and mechanisms that are in wide spread use. Many of them try to provide a generic infrastructure to be used by many different application layer protocols. The motivation behind the transport subsystem is to leverage these protocols where it seems useful.

There are a number of challenges to be addressed to map the security

provided by a secure transport into the SNMP architecture so that SNMP continues to work without any surprises. These challenges are discussed in detail in this document. For some key issues, design choices are discussed that may be made to provide a workable solution that meets operational requirements and fits into the SNMP architecture defined in [RFC3411].

3. Requirements of a Transport Model

3.1. Message Security Requirements

Transport security protocols SHOULD ideally provide the protection against the following message-oriented threats [RFC3411]:

1. modification of information
2. masquerade
3. message stream modification
4. disclosure

According to [RFC3411], it is not required to protect against denial of service or traffic analysis.

3.1.1. Security Protocol Requirements

There are a number of standard protocols that could be proposed as possible solutions within the transport subsystem. Some factors should be considered when selecting a protocol.

Using a protocol in a manner for which it was not designed has numerous problems. The advertised security characteristics of a protocol may depend on its being used as designed; when used in other ways, it may not deliver the expected security characteristics. It is recommended that any proposed model include a discussion of the applicability of the transport model.

A transport model should require no modifications to the underlying protocol. Modifying the protocol may change its security characteristics in ways that would impact other existing usages. If a change is necessary, the change should be an extension that has no impact on the existing usages. It is recommended that any transport model include a discussion of potential impact on other usages of the protocol.

It has been a long-standing requirement that SNMP be able to work when the network is unstable, to enable network troubleshooting and repair. The UDP approach has been considered to meet that need well, with an assumption that getting small messages through, even if out of order, is better than getting no messages through. There has been

a long debate about whether UDP actually offers better support than TCP when the underlying IP or lower layers are unstable. There has been recent discussion of whether operators actually use SNMP to troubleshoot and repair unstable networks.

There has been discussion of ways SNMP could be extended to better support management/monitoring needs when a network is running just fine. Use of a TCP transport, for example, could enable larger message sizes and more efficient table retrievals.

Transport models MUST be able to coexist with other transport models, and may be designed to utilize either TCP or UDP or SCTP.

3.2. SNMP Requirements

3.2.1. Architectural Modularity Requirements

SNMP version 3 (SNMPv3) is based on a modular architecture (described in [RFC3411] section 3) to allow the evolution of the SNMP protocol standards over time, and to minimize side effects between subsystems when changes are made.

The RFC3411 architecture includes a security subsystem for enabling different methods of providing security services, a messaging subsystem permitting different message versions to be handled by a single engine, an application subsystem to support different types of application processors, and an access control subsystem for allowing multiple approaches to access control. The RFC3411 architecture does not include a subsystem for transport models, despite the fact there are multiple transport mappings already defined for SNMP. This document addresses the need for a transport subsystem compatible with the RFC3411 architecture.

In SNMPv2, there were many problems of side effects between subsystems caused by the manipulation of MIB objects, especially those related to authentication and authorization, because many of the parameters were stored in shared MIB objects, and different models and protocols could assign different values to the objects. Contributors assumed slightly different shades of meaning depending on the models and protocols being used. As the shared MIB module design was modified to accommodate a specific model, other models which used the same MIB objects would be broken.

Abstract Service Interfaces (ASIs) were developed to pass model-independent parameters. The models were required to translate from their model-dependent formats into a model-independent format, defined using model-independent semantics, which would not impact other models.

Parameters have been provided in the ASIs to pass model-independent information about the authentication that has been provided. These parameters include a model-independent identifier of the security "principal", the security model used to perform the authentication, and which SNMP-specific security features were applied to the message (authentication and/or privacy).

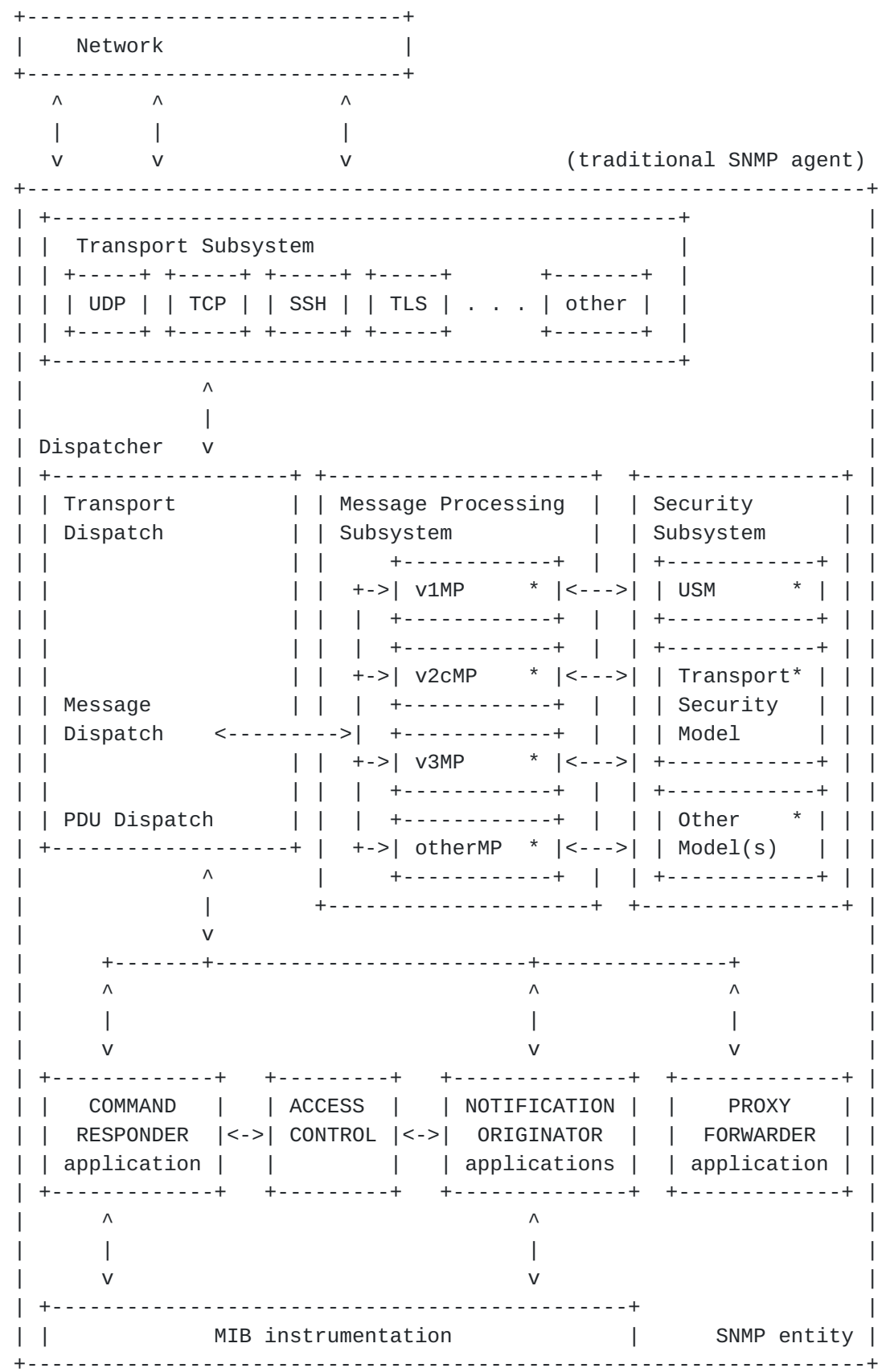
Parameters have been provided in the ASIs to pass model-independent transport address information. These parameters utilize the transportDomain and transportAddress

The design of a transport subsystem must abide the goals of the RFC3411 architecture defined in [RFC3411]. To that end, this transport subsystem proposal uses a modular design that will permit transport models to be advanced through the standards process independently of other transport models, and independent of other modular SNMP components as much as possible.

IETF standards typically require one mandatory to implement solution, with the capability of adding new mechanisms in the future. Part of the motivstion of developing transport models is to develop support for secure transport protocols, such as a transport model that utilizes the Secure Shell protocol. Any transport model should define one minimum-compliance security mechanism, preferably one which is already widely used to secure the transport layer protocol.

The Transport Subsystem permits multiple transport protocols to be "plugged into" the RFC3411 architecture, supported by corresponding transport models, including models that are security-aware.

The RFC3411 architecture,and the USM assume that a security model is called by a message-processing model and will perform multiple security functions within the security subsystem. A transport model that supports a secure transport protocol may perform similar security functions within the transport subsystem. A transport model may perform the translation of transport security parameters to/from security-model-independent parameters. To accommodate this, the ASIs for the transport subsystem, the messaging subsystem, and the security subsystem will be extended to pass security-model-independent values, and a cache of transport-specific information.


Network
(traditional SNMP agent)
Transport Subsystem
UDP
TCP
SSH
TLS
. . .
other
Dispatcher
Transport Dispatch
Message Dispatch
PDU Dispatch
Message Processing Subsystem
v1MP
v2cMP
v3MP
otherMP
Security Subsystem
USM
Transport Security Model
Other Model(s)
COMMAND RESPONDER application
ACCESS CONTROL
NOTIFICATION ORIGINATOR applications
PROXY FORWARDER application
MIB instrumentation
SNMP entity

3.2.1.1. USM and the RFC3411 Architecture

The following diagrams illustrate the difference in the security processing done by the USM model and the security processing potentially done by a transport model.

The USM security model is encapsulated by the messaging model, because the messaging model needs to perform the following steps (for incoming messages)

1) decode the ASN.1 (messaging model)
2) determine the SNMP security model and parameters (messaging model)
3) decrypt the encrypted portions of the message (security model)
4) translate parameters to model-independent parameters (security model)
5) determine which application should get the decrypted portions (messaging model), and
6) pass on the decrypted portions with model-independent parameters.

The USM approach uses SNMP-specific message security and parameters.

3.2.1.2. Transport Subsystem and the RFC3411 Architecture

With the Transport Subsystem, the order of the steps may differ and may be handled by different subsystems:

1) decrypt the encrypted portions of the message (transport layer)
2*) translate parameters to model-independent parameters (transport model)
3) determine the SNMP security model and parameters (transport model)
4) decode the ASN.1 (messaging model)
5) determine which application should get the decrypted portions (messaging model)
7) pass on the decrypted portions with model-independent security parameters

If a message is secured using non-SNMP-specific message security and parameters, then the transport model should provide the translation from the authenticated identity (e.g., an SSH user name) to the securityName in step 3.

3.2.1.3. Passing Information between Engines

A secure transport model will establish an encrypted tunnel between the transport models of two SNMP engines. One transport model instance encrypts all messages, and the other transport model instance decrypts the messages.

After a transport layer tunnel is established, then SNMP messages can conceptually be sent through the tunnel from one SNMP engine to

another SNMP engine. Once the tunnel is established, multiple SNMP messages may be able to be passed through the same tunnel.

3.2.2. Access Control Requirements

3.2.2.1. securityName Binding

For SNMP access control to function properly, security processing must establish a securityModel identifier, a securityLevel, and a securityName, which is the security model independent identifier for a principal. The message processing subsystem relies on a security model, such as USM, to play a role in security that goes beyond protecting the message - it provides a mapping between the USM-specific principal to a security-model independent securityName which can be used for subsequent processing, such as for access control.

The securityName MUST be bound to the mechanism-specific authenticated identity, and this mapping MUST be done for incoming messages before the security model passes securityName to the message processing model via the processIncoming() ASI. This translation from a mechanism-specific authenticated identity to a securityName MAY be done by the transport model, and the securityname is then provided to the security model to be passed to the message processing model.

If the type of authentication provided by the transport layer (e.g. TLS) is considered adequate to secure and/or encrypt the message, but inadequate to provide the desired granularity of access control (e.g. user-based), then a second authentication (e.g., one provided via a RADIUS server) MAY be used to provide the authentication identity which is bound to the securityName. This approach would require a good analysis of the potential for man-in-the-middle attacks or masquerade possibilities.

3.2.2.2. Separation of Authentication and Authorization

A transport model that provides security services should take care to not violate the separation of authentication and authorization in the RFC3411 architecture. The isAccessAllowed() primitive is used for passing security-model independent parameters between the subsystems of the architecture.

Mapping of (securityModel, securityName) to an access control policy should be handled within the access control subsystem, not the transport or security subsystems, to be consistent with the modularity of the RFC3411 architecture. This separation was a deliberate decision of the SNMPv3 WG, to allow support for authentication protocols which did not provide authorization

capabilities, and to support authorization schemes, such as VACM, that do not perform their own authentication.

An authorization model (in the access control subsystem) MAY require authentication by certain securityModels and a minimum securityLevel to allow access to the data.

Transport models that provide secure transport are an enhancement for the SNMPv3 privacy and authentication, but they are not a significant improvement for the authorization (access control) needs of SNMPv3. Only the model-independent parameters for the isAccessAllowed() primitive [RFC3411] are provided by the transport and security subsystems.

A transport model must not specify how the securityModel and securityName could be dynamically mapped to an access control mechanism, such as a VACM-style groupName. The mapping of (securityModel, securityName) to a groupName is a VACM-specific mechanism for naming an access control policy, and for tying the named policy to the addressing capabilities of the data modeling language (e.g. SMIv2 [RFC2578]), the operations supported, and other factors. Providing a binding outside the Access Control subsystem might create dependencies that could make it harder to develop alternate models of access control, such as one built on UNIX groups or Windows domains. The preferred approach is to pass the model-independent security parameters via the isAccessAllowed() ASI, and perform the mapping from the model-independent security parameters to an authorization-model-dependent access policy within the access control model.

To provide support for protocols which simultaneously send information for authentication and authorization, such as RADIUS [RFC2865], model-specific authorization information MAY be cached or otherwise made available to the access control subsystem, e.g., via a MIB table similar to the vacmSecurityToGroupTable, so the access control subsystem can create an appropriate binding between the model-independent securityModel and securityName and a model-specific access control policy. This may be highly undesirable, however, if it creates a dependency between a transport model or a security model and an access control model, just as it is undesirable for a transport model to create a dependency between an SNMP message version and the security provided by a transport model.

3.2.3. Security Parameter Passing Requirements

RFC3411 section 4 describes primitives to describe the abstract data flows between the various subsystems, models and applications within the architecture. Abstract Service Interfaces describe the flow of

data between subsystems within an engine. The ASIs generally pass model-independent information.

Within an engine using a 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.

The security parameters include a model-independent identifier of the security "principal", the security model used to perform the authentication, and which SNMP-specific security services were (should be) applied to the message (authentication and/or privacy).

In the RFC3411 architecture, which reflects the USM security model design, the messaging model must unpack SNMP-specific security parameters from an incoming message before calling a specific security model to authenticate and decrypt an incoming message, perform integrity checking, and translate model-specific security parameters into model-independent parameters.

When using a secure transport model, security parameters MAY be provided through means other than carrying them in the SNMP message. The parameters MAY be provided by SNMP applications for outgoing messages, and the parameters for incoming messages MAY be extracted from the transport layer by the transport model before the message is passed to the message processing subsystem.

For outgoing messages, even when a secure transport model will provide the security services, it is necessary to have an security model because it is the security model that actually creates the message from its component parts. Whether there are any security services provided by the security model for an outgoing message is model-dependent.

For incoming messages, even when a secure transport model provides security services, a security model is necessary because there might be some security functionality that can only be provided after the message version is known. The message version is determined by the Message Processing model and passed to the security model via the processIncoming() ASI.

The RFC3411 architecture has no ASI parameters for passing security information between a transport mapping (a transport model) and the dispatcher, and between the dispatcher and the message processing model.

This document describes a cache mechanism, into which the transport model puts information about the transport and security parameters

applied to a transport connection or an incoming message, and a security model MAY extract that information from the cache. A tmStateReference is passed as an extra parameter in the ASIs of the transport subsystem and the messaging and security subsystems, to identify the relevant cache.

This approach of passing a model-independent reference is consistent with the securityStateReference cache already being passed around in the RFC3411 ASIs.

3.3. Session Requirements

Some secure transports may have a notion of sessions, while other secure transports might provide channels or other session-like thing. Throughout this document, the term session is used in a broad sense to cover sessions, channels, and session-like things. Session refers to an association between two SNMP engines that permits the transmission of one or more SNMP messages within the lifetime of the session. How the session is actually established, opened, closed, or maintained is specific to a particular transport model.

Sessions are not part of the SNMP architecture described in [RFC3411], but are considered desirable because the cost of authentication can be amortized over potentially many transactions.

It is important to note that the architecture described in [RFC3411] does not include a session selector in the Abstract Service Interfaces, and neither is that done for the transport subsystem, so an SNMP application cannot select the session except by passing a unique combination of transport type, transport address, securityName, securityModel, and securityLevel.

All transport models should discuss the impact of sessions on SNMP usage, including how to establish/open a transport session (i.e., how it maps to the concepts of session-like things of the underlying protocol), how to behave when a session cannot be established, how to close a session properly, how to behave when a session is closed improperly, the session security properties, session establishment overhead, and session maintenance overhead.

To reduce redundancy, this document will discuss aspects that are expected to be common to all transport model sessions.

3.3.1. Session Establishment Requirements

SNMP applications must provide the transport type, transport address, securityName, securityModel, and securityLevel to be used for a session.

SNMP Applications typically have no knowledge of whether the session that will be used to carry commands was initially established as a notification session, or a request-response session, and SHOULD NOT make any assumptions based on knowing the direction of the session. If an administrator or transport model designer wants to differentiate a session established for different purposes, such as a notification session versus a request-response session, the application can use different securityNames or transport addresses (e.g., port 161 vs. port 162) for different purposes.

An SNMP engine containing an application that initiates communication, e.g., a Command Generator or Notification Originator, MUST be able to attempt to establish a session for delivery if a session does not yet exist. If a session cannot be established then the message is discarded.

Sessions are usually established by the transport model when no appropriate session is found for an outgoing message, but sessions may be established in advance to support features such as notifications. How sessions are established in advance is beyond the scope of this document.

Sessions are initiated by notification originators when there is no currently established connection that can be used to send the notification. For a client-server security protocol, this may require provisioning authentication credentials on the agent, either statically or dynamically, so the client/agent can successfully authenticate to a notification receiver.

A transport model must be able to determine whether a session does or does not exist, and must be able to determine which session has the appropriate security characteristics (transport type, transport address, securityName, securityModel, and securityLevel) for an outgoing message.

A transport model implementation MAY reuse an already established session with the appropriate transport type, transport address, securityName, securityModel, and securityLevel characteristics for delivery of a message originated by a different type of application than originally caused the session to be created. For example, an implementation that has an existing session originally established to receive a request may use that session to send an outgoing notification, and may use a session that was originally established to send a notification to send a request. Responses are expected to be returned using the same session that carried the corresponding request message. Reuse of sessions is not required for conformance.

If a session can be reused for a different type of message, but a

receiver is not prepared to accept different message types over the same session, then the message MAY be dropped by the receiver. This may strongly affect the usefulness of session reuse, and transport models should define a standard behavior for this circumstance.

3.3.2. Session Maintenance Requirements

A transport model can tear down sessions as needed. It may be necessary for some implementations to tear down sessions as the result of resource constraints, for example.

The decision to tear down a session is implementation-dependent. While it is possible for an implementation to automatically tear down each session once an operation has completed, this is not recommended for anticipated performance reasons. How an implementation determines that an operation has completed, including all potential error paths, is implementation-dependent.

The elements of procedure may discuss when cached information can be discarded, and the timing of cache cleanup may have security implications, but cache memory management is an implementation issue.

If a transport model defines MIB module objects to maintain session state information, then the transport model MUST describe what happens to the objects when a related session is torn down, since this will impact interoperability of the MIB module.

3.3.3. Message security versus session security

A transport model session is associated with state information that is maintained for its lifetime. This state information allows for the application of various security services to multiple messages. Cryptographic keys established at the beginning of the session SHOULD be used to provide authentication, integrity checking, and encryption services for data that is communicated during the session. The cryptographic protocols used to establish keys for a transport model session SHOULD ensure that fresh new session keys are generated for each session. If each session uses new session keys, then messages cannot be replayed from one session to another. In addition sequence information MAY be maintained in the session which can be used to prevent the replay and reordering of messages within a session.

A transport model session will typically have a single transport type, transport address, securityModel, securityName and securityLevel associated with it. If an exchange between communicating engines requires a different securityLevel or is on behalf of a different securityName, or uses a different securityModel, then another session would be needed. An immediate

consequence of this is that implementations should be able to maintain some reasonable number of concurrent sessions.

For transport models, securityName is typically specified during session setup, and associated with the session identifier.

SNMPv3 was designed to support multiple levels of security, selectable on a per-message basis by an SNMP application, because there is not much value in using encryption for a Commander Generator to poll for non-sensitive performance data on thousands of interfaces every ten minutes; the encryption may add significant overhead to processing of the messages.

Some transport models MAY support only specific authentication and encryption services, such as requiring all messages to be carried using both authentication and encryption, regardless of the security level requested by an SNMP application. A transport model MAY upgrade the requested security level, i.e. noAuth/noPriv and auth/noPriv MAY be sent over an authenticated and encrypted session.

4. Scenario Diagrams for the Transport Subsystem

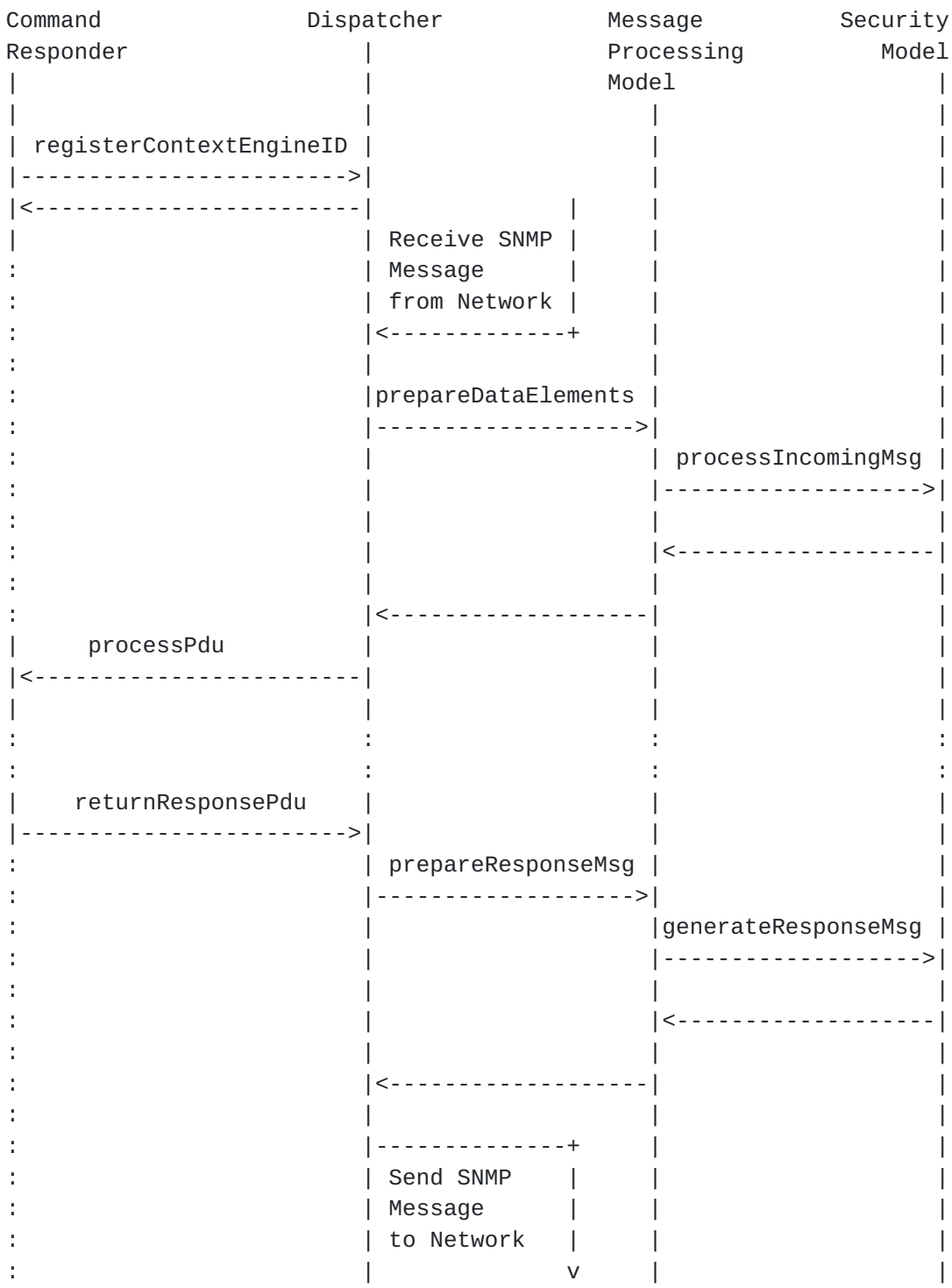
RFC3411 section 4.6 provides scenario diagrams to illustrate how an outgoing message is created, and how an incoming message is processed. Both diagrams are incomplete, however. In section 4.6.1, the diagram doesn't show the ASI for sending an SNMP request to the network or receiving an SNMP response message from the network. In section 4.6.2, the diagram doesn't illustrate the interfaces required to receive an SNMP message from the network, or to send an SNMP message to the network.

4.1. Command Generator or Notification Originator

This diagram from RFC3411 4.6.1 shows how a Command Generator or Notification Originator application [RFC3413] requests that a PDU be sent, and how the response is returned (asynchronously) to that application.

4.2. Command Responder

This diagram shows how a Command Responder or Notification Receiver application registers for handling a pduType, how a PDU is dispatched to the application after an SNMP message is received, and how the Response is (asynchronously) send back to the network.



5. 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. 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.

5.1. 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 (transportDomain, 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.

5.2. tmStateReference

For each message or transport session, information about the message security is stored in a cache, which may inlcude model- and mechanism-specific parameters. The tmStateReference is passed between subsystems to provide a handle for the cache. A transport model may store transport-specific parameters in the cache for subsequent usage. Since the contents of a cache are meaningful only within an implementation, and not on-the-wire, the format of the cache is implementation-specific.

The state referenced by tmStateReference may be saved in a Local Configuration Datastore (LCD) to make it available across multiple messages, as compared to securityStateReference which is designed to be saved only for the life of a request-response pair of messages. It is expected that an LCD will allow lookup based on the combination of transportDomain, transportAddress, securityName, securityModel, and securityLevel, and that the cache contain these values to reference entries in the LCD.

6. Abstract Service Interfaces

Abstract service interfaces have been defined by RFC 3411 to describe the conceptual data flows between the various subsystems within an SNMP entity.

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, and if state information is available when a session is closed, the session state information should also be released.

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. 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
OUT  tmStateReference        -- (NEW) reference to transport state
            )
```

Note that tmStateReference has been added to this ASI.

The IN parameters of the prepareOutgoingMessage() ASI are used to pass information from the dispatcher (from 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().

The abstract service primitive from a Message Processing Model to a Security Model to generate the components of a Response message is generateResponseMsg().

Upon completion of processing, the Security Model returns statusInformation. If the process was successful, the completed message is returned. If the process was not successful, then an errorIndication is returned.

The OUT parameters of the prepareOutgoingMessage() ASI are used to pass information from the message processing model to the dispatcher and on to the transport model:

6.2. Processing for an Outgoing Message

The sendMessage ASI is used to pass a message from the Dispatcher to the appropriate transport model for sending.

```
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             -- reference to transport state
 )
```

6.3. Processing an Incoming SNMP Message

6.3.1. Processing an Incoming Message

If one does not exist, the Transport Model will need to create an entry in a Local Configuration Datastore referenced by tmStateReference. This information will include transportDomain, transportAddress, the securityModel, the securityLevel, and the securityName, plus any model or mechanism-specific details. How this information is determined is model-specific.

The recvMessage ASI is used to pass a message from the transport subsystem to the Dispatcher.

```
statusInformation =
recvMessage(
IN   transportDomain              -- origin transport domain
IN   transportAddress             -- origin transport address
IN   incomingMessage              -- the message received
IN   incomingMessageLength        -- its length
IN   tmStateReference             -- reference to transport state
 )
```

6.3.2. 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           -- (NEW) 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.

6.3.3. Processing an Incoming Message

This section describes the procedure followed by the 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          -- (NEW) 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) The securityEngineID is set to a value in a model-specific manner. If the securityEngineID is not utilized by the specific model, then it should be set to the local snmpEngineID, to satisfy the SNMPv3 message processing model in RFC 3412 section 7.2 13a).

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

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.

6) 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.

7. Security Considerations

This document describes an architectural approach that would permit SNMP to utilize transport layer security services. Each proposed transport model should discuss the security considerations of the transport model.

It is considered desirable by some industry segments that SNMP

transport models should utilize transport layer security that addresses perfect forward secrecy at least for encryption keys. Perfect forward secrecy guarantees that compromise of long term secret keys does not result in disclosure of past session keys. The editors recommend that each proposed transport model include a discussion in its security considerations of whether perfect forward security is appropriate for the transport model.

Since the cache and LCD will contain security-related parameters, they should be kept in protected storage.

8. IANA Considerations

This document requires no action by IANA.

9. Acknowledgments

The Integrated Security for SNMP WG would like to thank the following people for their contributions to the process:

The authors of submitted security model proposals: Chris Elliot, Wes Hardaker, Dave Harrington, Keith McCloghrie, Kaushik Narayan, Dave Perkins, Joseph Salowey, and Juergen Schoenwaelder.

The members of the Protocol Evaluation Team: Uri Blumenthal, Lakshminath Dondeti, Randy Presuhn, and Eric Rescorla.

WG members who committed to and performed detailed reviews: Jeffrey Hutzelman

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.

[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.

[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.

[RFC3417] Presuhn, R., "Transport Mappings for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3417, December 2002.

10.2. Informative References

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

[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.

[RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, April 2006.

[RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.

[RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

[I-D.ietf-netconf-ssh] Wasserman, M. and T. Goddard, "Using the NETCONF Configuration Protocol over Secure Shell (SSH)", draft-ietf-netconf-ssh-06 (work in progress), March 2006.

Appendix A. Parameter Table

Following is a CSV formatted matrix useful for tracking data flows into and out of the dispatcher, transport, message, and security subsystems. Import this into your favorite spreadsheet or other CSV compatible application. You will need to remove lines feeds from the second, third, and fourth lines, which needed to be wrapped to fit into RFC limits.

A.1. ParameterList.csv

```
,Dispatcher,,,,Messaging,,,Security,,,Transport,

,sendPDU,returnResponse,processPDU,processResponse,

prepareOutgoingMessage,prepareResponseMessage,prepareDataElements,

generateRequest,processIncoming,generateResponse,

sendMessage,recvMessage

transportDomain,In,,,,In,,In,,,,,In

transportAddress,In,,,,In,,In,,,,,In

destTransportDomain,,,,,Out,Out,,,,,In,

destTransportAddress,,,,,Out,Out,,,,,In,

messageProcessingModel,In,In,In,In,In,In,In,Out,In,In,In,,

securityModel,In,In,In,In,In,In,In,Out,In,In,In,,

securityName,In,In,In,In,In,In,In,Out,In,Out,In,,

securityLevel,In,In,In,In,In,In,In,Out,In,In,In,,

contextEngineID,In,In,In,In,In,In,In,Out,,,,,

contextName,In,In,In,In,In,In,In,Out,,,,,

expectResponse,In,,,,In,,,,,,,

PDU,In,In,In,In,In,In,In,Out,,,,,

pduVersion,In,In,In,In,In,In,In,Out,,,,,

statusInfo,Out,In,,In,,In,Out,Out,Out,Out,,
```

errorIndication,Out,Out,,,,,Out,,,,,

sendPduHandle,Out,,,In,In,,Out,,,,,

maxSizeResponsePDU,,In,In,,,In,Out,,Out,,,

stateReference,,In,In,,,In,Out,,,,,

wholeMessage,,,,,Out,Out,In,Out,In,Out,In,In

messageLength,,,,,Out,Out,In,Out,In,Out,In,In

maxMessageSize,,,,,,,,In,In,In,,

globalData,,,,,,,,In,,In,,

securityEngineID,,,,,,,,In,Out,In,,

scopedPDU,,,,,,,,In,Out,In,,

securityParameters,,,,,,,,Out,In,Out,,

securityStateReference,,,,,,,,,Out,In,,

pduType,,,,,,,Out,,,,,

tmStateReference,,,,,Out,Out,In,,In,,In,In

Appendix B. Why tmStateReference?

This appendix considers why a cache-based approach was selected for passing parameters. This section may be removed from subsequent revisions of the document.

There are four approaches that could be used for passing information between the Transport Model and an Security Model.

1. one could define an ASI to supplement the existing ASIs, or
2. one could add a header to encapsulate the SNMP message,
3. one could utilize fields already defined in the existing SNMPv3 message, or
4. one could pass the information in an implementation-specific cache or via a MIB module.

B.1. Define an Abstract Service Interface

Abstract Service Interfaces (ASIs) [RFC3411] are defined by a set of primitives that specify the services provided and the abstract data

elements that are to be passed when the services are invoked. Defining additional ASIs to pass the security and transport information from the transport subsystem to security subsystem has the advantage of being consistent with existing RFC3411/3412 practice, and helps to ensure that any transport model proposals pass the necessary data, and do not cause side effects by creating model-specific dependencies between itself and other models or other subsystems other than those that are clearly defined by an ASI.

B.2. Using an Encapsulating Header

A header could encapsulate the SNMP message to pass necessary information from the Transport Model to the dispatcher and then to a messaging security model. The message header would be included in the wholeMessage ASI parameter, and would be removed by a corresponding messaging model. This would imply the (one and only) messaging dispatcher would need to be modified to determine which SNMP message version was involved, and a new message processing model would need to be developed that knew how to extract the header from the message and pass it to the Security Model.

B.3. Modifying Existing Fields in an SNMP Message

[RFC3412] describes the SNMPv3 message, which contains fields to pass security related parameters. The transport subsystem could use these fields in an SNMPv3 message, or comparable fields in other message formats to pass information between transport models in different SNMP engines, and to pass information between a transport model and a corresponding messaging security model.

If the fields in an incoming SNMPv3 message are changed by the Transport Model before passing it to the Security Model, then the Transport Model will need to decode the ASN.1 message, modify the fields, and re-encode the message in ASN.1 before passing the message on to the message dispatcher or to the transport layer. This would require an intimate knowledge of the message format and message versions so the Transport Model knew which fields could be modified. This would seriously violate the modularity of the architecture.

B.4. Using a Cache

This document describes a cache, into which the Transport Model puts information about the security applied to an incoming message, and a Security Model can extract that information from the cache. Given that there may be multiple TM-security caches, a tmStateReference is passed as an extra parameter in the ASIs between the transport subsystem and the security subsystem, so the Security Model knows which cache of information to consult.

This approach does create dependencies between a specific Transport Model and a corresponding specific Security Model. However, the approach of passing a model-independent reference to a model-dependent cache is consistent with the securityStateReference already being passed around in the RFC3411 ASIs.

Appendix C. Open Issues

Appendix D. Change Log

NOTE to RFC editor: Please remove this change log before publishing this document as an RFC.

Changes from revision -04- to -05-

removed all objects from the MIB module.
changed document status to "Standard" rather than the xml2rfc default of informational.

changed mention of MD5 to SHA
moved addressing style to TDomain and TAddress
modified the diagrams as requested
removed the "layered stack" diagrams that compared USM and a Transport Model processing
removed discussion of speculative features that might exist in future transport models
removed openSession() and closeSession() ASIs, since those are model-dependent
removed the MIB module
removed the MIB boilerplate into (this memo defines a SMIv2 MIB ...)
removed IANA considerations related to the now-gone MIB module
removed security considerations related to the MIB module
removed references needed for the MIB module
changed recvMessage ASI to use origin transport domain/address
updated Parameter CSV appendix

Changes from revision -03- to -04-

changed title from Transport Mapping Security Model Architectural Extension to Transport Subsystem
modified the abstract and introduction
changed TMSM to TMS
changed MPSP to simply Security Model
changed SMSP to simply Security Model
changed TMSP to Transport Model
removed MPSP and TMSP and SMSP from Acronyms section

modified diagrams
removed most references to dispatcher functionality
worked to remove dependencies between transport and security models.
defined snmpTransportModel enumeration similar to snmpSecurityModel, etc.
eliminated all reference to SNMPv3 msgXXXX fields
changed tmSessionReference back to tmStateReference

Changes from revision -02- to -03-

- removed session table from MIB module
- removed sessionID from ASIs
- reorganized to put ASI discussions in EOP section, as was done in SSHSM
- changed user auth to client auth
- changed tmStateReference to tmSessionReference
- modified document to meet consensus positions published by JS
-
  - authoritative is model-specific
  - msgSecurityParameters usage is model-specific
  - msgFlags vs. securityLevel is model/implementation-specific
  - notifications must be able to cause creation of a session
  - security considerations must be model-specific
  - TDomain and TAddress are model-specific
  - MPSP changed to SMSP (Security model security processing)

Changes from revision -01- to -02-

- wrote text for session establishment requirements section.
- wrote text for session maintenance requirements section.
- removed section on relation to SNMPv2-MIB
- updated MIB module to pass smilint
- Added Structure of the MIB module, and other expected MIB-related sections.
- updated author address
- corrected spelling
- removed msgFlags appendix
- Removed section on implementation considerations.
- started modifying the security boilerplate to address TMS and MIB security issues
- reorganized slightly to better separate requirements from proposed solution. This probably needs additional work.
- removed section with sample protocols and sample tmSessionReference.
- Added section for acronyms

- moved section comparing parameter passing techniques to appendix.
- Removed section on notification requirements.

Changes from revision -00-
- changed SSH references from I-Ds to RFCs
- removed parameters from tmSessionReference for DTLS that revealed lower layer info.
- Added TMS-MIB module
- Added Internet-Standard Management Framework boilerplate
- Added Structure of the MIB Module
- Added MIB security considerations boilerplate (to be completed)
- Added IANA Considerations
- Added ASI Parameter table
- Added discussion of Sessions
- Added Open issues and Change Log
- Rearranged sections

Authors' Addresses

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

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

Juergen Schoenwaelder
International University Bremen
Campus Ring 1
28725 Bremen
Germany

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

Full Copyright Statement

Intellectual Property

Acknowledgement

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