

Network Working Group
Internet-Draft
Updates: [3411](#), 3412, 3414, 3417
(if approved)
Intended status: Standards Track
Expires: November 2, 2007

D. Harrington
Huawei Technologies (USA)
J. Schoenwaelder
International University Bremen
May 1, 2007

Transport Subsystem for the Simple Network Management Protocol (SNMP)
draft-ietf-isms-tmsm-08

Status of This Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with [Section 6 of BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/1id-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on November 2, 2007.

Copyright Notice

Copyright (C) The IETF Trust (2007).

Abstract

This document defines a Transport Subsystem, extending the Simple Network Management Protocol (SNMP) architecture defined in [RFC 3411](#). This document defines 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 describes some key aspects that need to be considered for any Transport Model for SNMP.

Table of Contents

1.	Introduction	3
1.1.	The Internet-Standard Management Framework	3
1.2.	Where this Extension Fits	3
1.3.	Conventions	5
2.	Motivation	5
3.	Requirements of a Transport Model	7
3.1.	Message Security Requirements	7
3.1.1.	Security Protocol Requirements	7
3.2.	SNMP Requirements	8
3.2.1.	Architectural Modularity Requirements	8
3.2.2.	Access Control Requirements	12
3.2.3.	Security Parameter Passing Requirements	13
3.2.4.	Separation of Authentication and Authorization	14
3.3.	Session Requirements	15
3.3.1.	Session Establishment Requirements	15
3.3.2.	Session Maintenance Requirements	16
3.3.3.	Message security versus session security	16
4.	Scenario Diagrams and the Transport Subsystem	17
5.	Cached Information and References	18
5.1.	securityStateReference	18
5.2.	tmStateReference	18
6.	Abstract Service Interfaces	19
6.1.	sendMessage ASI	19
6.2.	Other Outgoing ASIs	20
6.3.	The receiveMessage ASI	21
6.4.	Other Incoming ASIs	22
7.	Security Considerations	23
8.	IANA Considerations	24
9.	Acknowledgments	24
10.	References	24
10.1.	Normative References	24
10.2.	Informative References	25
Appendix A.	Parameter Table	26
A.1.	ParameterList.csv	26
Appendix B.	Why tmStateReference?	28
B.1.	Define an Abstract Service Interface	28
B.2.	Using an Encapsulating Header	28
B.3.	Modifying Existing Fields in an SNMP Message	29
B.4.	Using a Cache	29
Appendix C.	Open Issues	29
Appendix D.	Change Log	30

1. Introduction

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

1.1. The Internet-Standard Management Framework

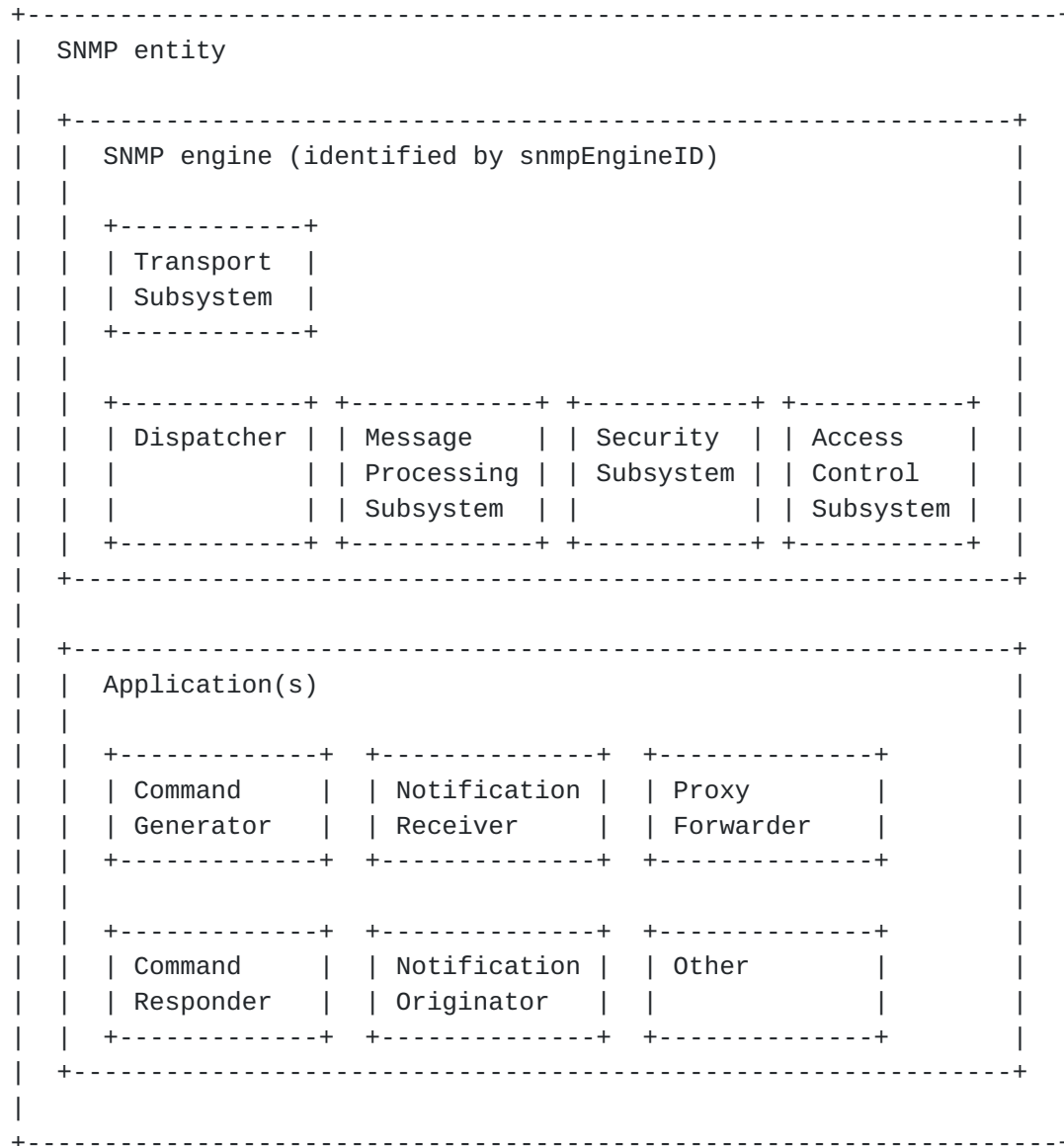
For a detailed overview of the documents that describe the current Internet-Standard Management Framework, please refer to [section 7 of RFC 3410](#) [[RFC3410](#)].

1.2. Where this Extension Fits

It is expected that readers of this document will have read [RFC3410](#) and [RFC3411](#), and have a general understanding of the functionality defined in RFCs 3412-3418.

The "Transport Subsystem" is an additional component for the SNMP Engine depicted in [RFC3411, section 3.1](#).

The following diagram depicts its place in the [RFC3411](#) architecture.:



The transport mappings defined in [RFC3417](#) do not provide lower-layer security functionality, and thus do not provide transport-specific security parameters. This document updates [RFC3411](#) and [RFC3417](#) by defining an architectural extension and ASIs that transport mappings (models) can use to pass transport-specific security parameters to other subsystems, including transport-specific security parameters translated into the transport-independent `securityName` and `securityLevel`.

1.3. 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](#)].

The key words "must", "must not", "required", "shall", "shall not", "should", "should not", "recommended", "may", and "optional" in this document are not to be interpreted as described in [RFC2119](#). They will usually, but not always, be used in a context relating to compatibility with the [RFC3411](#) architecture or the subsystem defined here, but which might have no impact on on-the-wire compatibility. These terms are used as guidance for designers of proposed IETF models to make the designs compatible with [RFC3411](#) subsystems and Abstract Service Interfaces (see [section 3.2](#)). Implementers are free to implement differently. Some usages of these lowercase terms are simply normal English usage.

2. Motivation

Just as there are multiple ways to secure one's home or business, in a continuum of alternatives, there are multiple ways to secure a network management protocol. Let's consider three general approaches.

In the first approach, an individual could sit on his front porch waiting for intruders. In the second approach, he could hire an employee, schedule the employee, position the employee to guard what he wants protected, hire a second guard to cover if the first gets sick, and so on. In the third approach, he could hire a security company, tell them what he wants protected, and they could hire employees, train them, position the guards, schedule the guards, send a replacement when a guard cannot make it, etc., thus providing the desired security, with no significant effort on his 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 (e.g., SHA), 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 to use

external mechanisms to handle the distribution of keys for use by USM. The more important issue is that operators wanted to leverage a single user base that wasn't specific to SNMP.

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 [[RFC4346](#)], 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 [[RFC4741](#)].

This document defines a Transport Subsystem extension to the [RFC3411](#) architecture based on the third approach. This extension specifies how other lower layer protocols with common security infrastructures can be used underneath the SNMP protocol and the desired goal of unified administrative security can be met.

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) [[RFC4346](#)] 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 provide interoperability with existing implementations. These challenges are described in detail in this document. For some key issues, design choices are described that might 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 provide protection against the following message-oriented threats [\[RFC3411\]](#):

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

These threats are described in [section 1.4 of \[RFC3411\]](#). It is not required to protect against denial of service or traffic analysis, but it should not make those threats significantly worse.

[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 might depend on it being used as designed; when used in other ways, it might not deliver the expected security characteristics. It is recommended that any proposed model include a description of the applicability of the Transport Model.

A Transport Model SHOULD require no modifications to the underlying protocol. Modifying the protocol might 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. Any Transport Model SHOULD include a description of potential impact on other usages of the protocol.

Transport Models MUST be able to coexist with each other.

3.2. SNMP Requirements

3.2.1. Architectural Modularity Requirements

SNMP version 3 (SNMPv3) is based on a modular architecture (defined 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 Message Processing Subsystem permitting different message versions to be handled by a single engine, Applications(s) 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. 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.

The design of this Transport Subsystem accepts the goals of the [RFC3411](#) architecture defined in [section 1.5 of \[RFC3411\]](#). This Transport Subsystem 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.

Parameters have been added to the ASIs to pass model-independent transport address information.

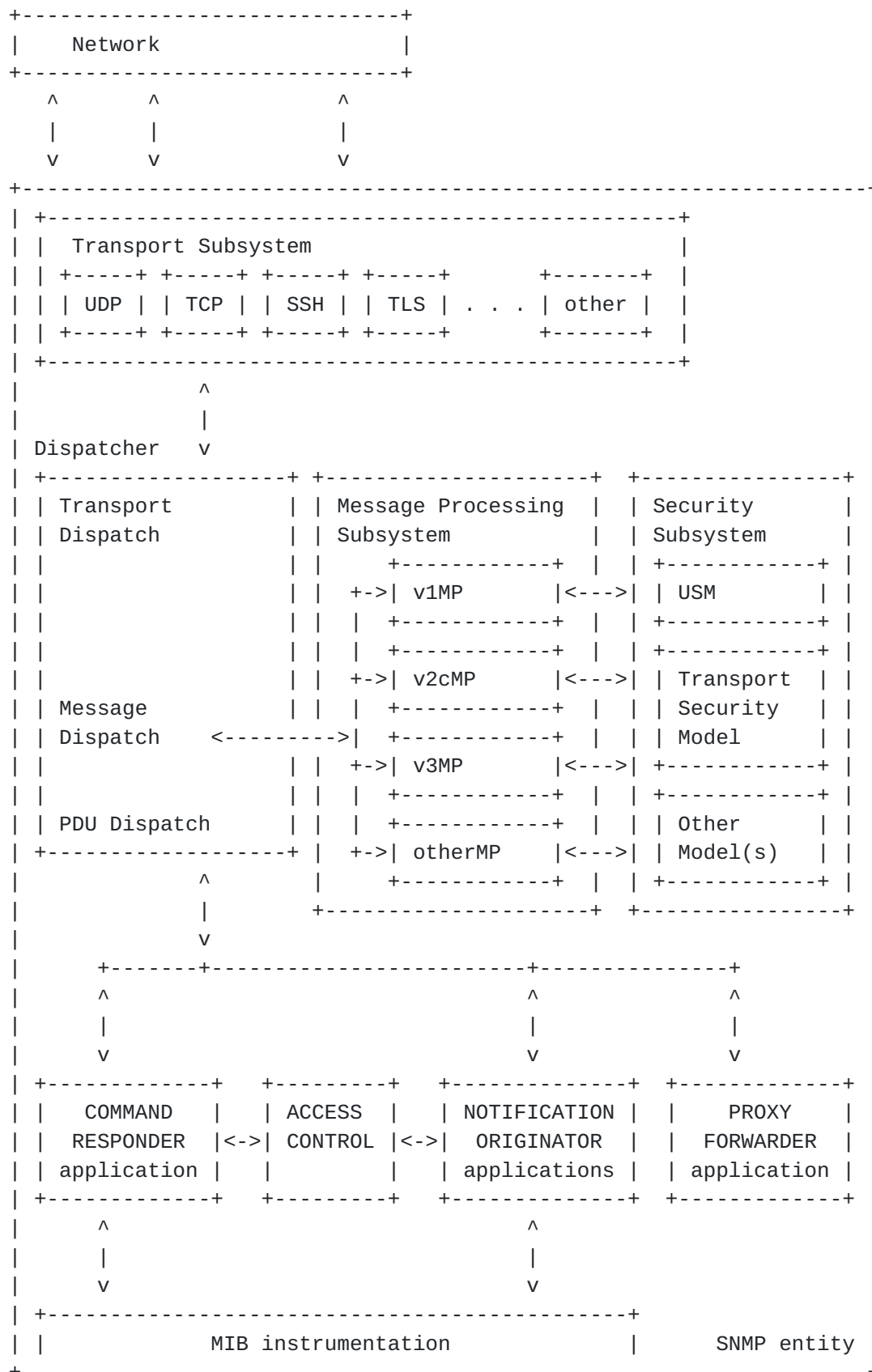
IETF standards typically require one mandatory to implement solution, with the capability of adding new mechanisms in the future. Part of the motivation of developing Transport Models is to develop support for secure transport protocols, such as a Transport Model that utilizes the Secure Shell protocol. Any Transport Model SHOULD define one minimum-compliance security mechanism, such as certificates, to ensure a basic level of interoperability, but should also be able to support additional existing and new mechanisms.

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 Security Subsystem 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 might perform similar security functions within the Transport Subsystem. A Transport Model might perform the translation of transport security parameters to/from security-model-independent parameters.

To accommodate this, an implementation-specific cache of transport-specific information will be described (not shown), and the data flows between the Transport Subsystem and the Transport Dispatch, between the Message Dispatch and the Message Processing Subsystem, and between the Message Processing Subsystem and the Security Subsystem will be extended to pass security-model-independent values. New Security Models may also be defined that understand how to work with the modified ASIs and the cache. One such Security Model, the Transport Security Model, is defined in [\[I-D.ietf-isms-transport-security-model\]](#)

The following diagram depicts the SNMPv3 architecture including the new Transport Subsystem defined in this document, and a new Transport Security Model defined in [\[I-D.ietf-isms-transport-security-model\]](#).



3.2.1.1. Processing Differences between USM and Secure Transport

USM and secure transports differ in the processing order and responsibilities within the [RFC3411](#) architecture. While the steps are the same, they occur in a different order, and may be done by different subsystems. The following lists illustrate the difference in the flow and the responsibility for different processing steps for incoming messages when using USM and when using a secure transport. (Note that these lists are simplified for illustrative purposes, and do not represent all details of processing. Transport Models must provide the detailed elements of procedure.)

With USM and other Security Models, security processing starts when the Message Processing Model decodes portions of the ASN.1 message to extract an opaque block of security parameters and header parameters that identify which Security Model should process the message to perform authentication, decryption, timeliness checking, integrity checking, and translation of parameters to model-independent parameters. A secure transport performs those security functions on the message, before the ASN.1 is decoded.

Step 6 cannot occur until after decryption occurs. Step 6 and beyond are the same for USM and a secure transport.

3.2.1.1.1. USM and the [RFC3411](#) Architecture

- 1) decode the ASN.1 header (Message Processing Model)
- 2) determine the SNMP Security Model and parameters (Message Processing Model)
- 3) verify securityLevel. [Security Model]
- 4) translate parameters to model-independent parameters (Security Model)
- 5) authenticate and decrypt message. [Security Model]
- 6) determine the pduType in the decrypted portions (Message Processing Model), and
- 7) pass on the decrypted portions with model-independent parameters.

3.2.1.2. Transport Subsystem and the [RFC3411](#) Architecture

- 1) authenticate and decrypt message. [Transport Model]
- 2) translate parameters to model-independent parameters (Transport Model)
- 3) decode the ASN.1 header (Message Processing Model)
- 4) determine the SNMP Security Model and parameters (Message Processing Model)
- 5) verify securityLevel [Security Model]
- 6) determine the pduType in the decrypted portions (Message Processing Model), and
- 7) pass on the decrypted portions with model-independent security parameters

If a message is secured using a secure transport layer, 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 authenticated and/or encrypted tunnel between the Transport Models of two SNMP engines. After a transport layer tunnel is established, then SNMP messages can be sent through the tunnel from one SNMP engine to the other SNMP engine. Transport Models MAY support sending multiple SNMP messages through the same tunnel.

3.2.2. Access Control Requirements

[RFC3411](#) made some design decisions related to the support of an Access Control Subsystem. These include a securityName and securityLevel mapping, the separation of Authentication and Authorization, and the passing of model-independent security parameters.

3.2.2.1. securityName and securityLevel Mapping

For SNMP access control to function properly, Security Models MUST establish 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 security-model-specific principal to a security-model independent securityName which can be used for subsequent

processing, such as for access control.

The securityName MUST be mapped from 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 might be done by the Transport Model, and the securityName is then provided to the Security Model via the tmStateReference to be passed to the Message Processing Model.

3.2.3. Security Parameter Passing Requirements

[RFC3411 section 4](#) describes abstract data flows between the subsystems, models and applications within the architecture. Abstract Service Interfaces describe the flow of data, passing model-independent information between subsystems within an engine. The [RFC3411](#) architecture has no ASI parameters for passing security information between the Transport Subsystem and the dispatcher, or between the dispatcher and the Message Processing Model. This document defines or modifies ASIs for this purpose.

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

A Message Processing Model might 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 security-model-specific parameters into model-independent parameters. When using a secure Transport Model, some security parameters might be provided through means other than carrying them in the SNMP message; some of the parameters for incoming messages might be extracted from the transport layer by the Transport Model before the message is passed to the Message Processing Subsystem.

This document describes a cache mechanism (see [Section 5](#)), 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 Message Processing 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.

For outgoing messages, even when a secure Transport Model will provide the security services, a Message Processing Model might have a Security Model actually create the message from its component parts. Whether there are any security services provided by the Security Model for an outgoing message is security-model-dependent. For incoming messages, even when a secure Transport Model provides security services, a Security Model might provide some security functionality that can only be provided after the message version or other parameters are extracted from the message.

3.2.4. Separation of Authentication and Authorization

The [RFC3411](#) architecture defines a separation of authentication and authorization (access control), and a Transport Model that provides security services should take care to not violate this separation. A Transport Model does not know which securityModel will be used for an incoming message, so must not specify how the securityModel and securityName could be dynamically mapped to an access control mechanism, such as a VACM-style groupName.

The RECOMMENDED 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 access-control-model-dependent policy within the Access Control Model. The isAccessAllowed ASI is used for passing the securityModel, securityName, and securityLevel parameters that are independent of any specific security model and any specific access control model to the Access Control Subsystem.

The mapping of (securityModel, securityName, securityLevel) to an access-control-model-specific policy should be handled within a specific access control model. This mapping should not be done in 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 (access control) capabilities, and to support authorization schemes, such as VACM, that do not perform their own authentication.

The View-based Access Control Model uses the securityModel and the securityName as inputs to check for access rights. It determines the groupName as a function of securityModel and securityName. 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.

3.3. Session Requirements

Some secure transports might have a notion of sessions, while other secure transports might provide channels or other session-like mechanism. Throughout this document, the term session is used in a broad sense to cover sessions, channels, and session-like mechanisms. 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 defined in [\[RFC3411\]](#), but are considered desirable because the cost of authentication can be amortized over potentially many transactions.

The architecture defined 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 has no mechanism to select a session using the ASIs except by passing a unique combination of transportDomain, transportAddress, securityName, and securityLevel. Implementers, of course, might provide non-standard mechanisms to select sessions. The transportDomain and transportAddress identify the transport connection to a remote network node; the securityName identifies which security principal to communicate with at that address (e.g., different NMS applications), and the securityLevel might permit selection of different sets of security properties for different purposes (e.g., encrypted SETs vs. non-encrypted GETs).

To reduce redundancy, this document describes aspects that are expected to be common to all Transport Model sessions.

3.3.1. Session Establishment Requirements

SNMP applications provide the transportDomain, transportAddress, securityName, and securityLevel to be used to identify a session in a transport-independent manner.

For an outgoing message, securityLevel is the requested security for the message, passed in the ASIs. If the Transport Model cannot provide at least the requested level of security, the Transport Model SHOULD discard the message and notify the dispatcher that sending the message failed.

A Transport Model determines whether an appropriate session exists (transportDomain, transportAddress, securityName, and securityLevel) for an outgoing message. If an appropriate session does not yet

exist, the Transport Model attempts to establish a session for delivery . If a session cannot be established then the message is discarded and the dispatcher should be notified that sending the message failed.

Depending on the secure transport protocol, session establishment might require provisioning authentication credentials on the agent, either statically or dynamically, so the client/agent can successfully authenticate to a receiver.

The Transport Subsystem has no knowledge of pduType, so cannot distinguish between a session created to carry different pduTypes. To differentiate a session established for different purposes, such as a notification session versus a request-response session, an application can use different securityNames or transport addresses. For example, in SNMPv1, UDP ports 161 and 162 were used to differentiate types of traffic. New transport models may define a single well-known port for all traffic types. Administrators might choose to define one port for SNMP traffic, but configure notifications to be sent to a different port.

3.3.2. Session Maintenance Requirements

A Transport Model can tear down sessions as needed. It might 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 describe when cached information can be discarded, in some circumstances, and the timing of cache cleanup might 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** define what **SHOULD** happen 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. In addition sequence information might be maintained in the session which can be used to prevent the replay and reordering of messages within a session. If each session uses new keys, then a cross-session replay attack will be unsuccessful; that is, an attacker cannot successfully replay on one session a message he observed from another session. A good security protocol will also protect against replay attacks within a session; that is, an attacker cannot successfully replay a message observed earlier in the same session.

A Transport Model session will have a single transportDomain, transportAddress, securityName and securityLevel associated with it. If an exchange between communicating engines requires a different securityLevel or is on behalf of a different securityName, 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 should be 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, for example, there is not much value in using encryption for a Commander Generator to poll for potentially non-sensitive performance data on thousands of interfaces every ten minutes; the encryption might add significant overhead to processing of the messages.

Some Transport Models might 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. noAuthNoPriv and authNoPriv MAY be sent over an authenticated and encrypted session.

4. Scenario Diagrams and the Transport Subsystem

[RFC3411 section 4.6.1](#) and 4.6.2 provide scenario diagrams to illustrate how an outgoing message is created, and how an incoming message is processed. [RFC3411](#) does not define ASIs for "Send SNMP Request Message to Network" or "Receive SNMP Response Message from Network", and does not define ASIs for "Receive SNMP Message from

Network" or "Send SNMP message to Network".

This document defines a sendMessage ASI to send SNMP messages to the network, regardless of pduType, and a receiveMessage ASI to receive SNMP messages from the network, regardless of pduType.

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 flows between subsystems.

There are two levels of state that might 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 description of the data that flows between subsystems. An implementation might 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

The securityStateReference parameter is defined in [RFC3411](#). securityStateReference is not accessible to models of the Transport Subsystem.

5.2. tmStateReference

For each transport session, information about the message security is stored in a cache to pass model- and mechanism-specific parameters. The state referenced by tmStateReference may be saved across multiple messages, in a Local Configuration Datastore (LCD), as compared to securityStateReference which is usually only saved for the life of a request-response pair of messages.

For security reasons, if a secure transport session is closed between the time a request message is received and the corresponding response message is sent, then the response message **MUST** be discarded, even if a new session has been established. The `tmStateReference` captured during processing of an incoming message **SHOULD** include a transport-specific session identifier. Each Security Model **SHOULD** pass a `tmSameSession` parameter in the `tmStateReference` cache for outgoing messages to indicate whether the same session must be used for the outgoing message as was used for the corresponding incoming message. If the session identified in the `tmStateReference` does not match the current established session, the message **MUST** be discarded, and the dispatcher should be notified the sending of the message failed.

Since the contents of a cache are meaningful only within an implementation, and not on-the-wire, the format of the cache and the LCD are implementation-specific.

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, and to help keep the subsystems independent of each other except for the common parameters.

This document follows the example of [RFC3411](#) regarding the release of state information, and regarding error indications.

1) The release of state information is not always explicitly specified in a transport model. 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. Note that keeping sensitive security information longer than necessary might introduce potential vulnerabilities to an implementation.

2) An error indication in `statusInformation` may include 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. `sendMessage` ASI

The `sendMessage` ASI is used to pass a message from the Dispatcher to the appropriate Transport Model for sending.

If present and valid, the `tmStateReference` refers to a cache

containing transport-model-specific parameters for the transport and transport security. How the information in the cache is used is transport-model-dependent and implementation-dependent. How a `tmStateReference` is determined to be present and valid is implementation-dependent.

This may sound underspecified, but keep in mind that a transport model might be something like SNMP over UDP over IPv6, where no security is provided, so it might have no mechanisms for utilizing a `securityName` and `securityLevel`.

```
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
)
```

[6.2.](#) Other Outgoing ASIs

A `tmStateReference` parameter has been added to the `prepareOutgoingMessage`, `generateRequestMsg`, and `generateResponseMsg` ASIs as an OUT parameter. The `transportDomain` and `transportAddress` parameters have been added to the `generateRequestMsg`, and `generateResponseMsg` ASIs as IN parameters (not shown).

```
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
```


)

The `tmStateReference` parameter of `generateRequestMsg` or `generateResponseMsg` is passed in the return parameters of the Security Subsystem to the Message Processing Subsystem. If a cache exists for a session identifiable from `transportDomain`, `transportAddress`, `securityModel`, `securityName`, and `securityLevel`, then an appropriate Security Model might create a `tmStateReference` to the cache and pass that as an OUT parameter.

If one does not exist, the Security Model might create a cache referenced by `tmStateReference`. This information might include `transportDomain`, `transportAddress`, the `securityLevel`, and the `securityName`, plus any model or mechanism-specific details. The contents of the cache may be incomplete until the Transport Model has established a session. What information is passed, and how this information is determined, is implementation and security-model-specific.

The `prepareOutgoingMessage` ASI passes `tmStateReference` from the Message Processing Subsystem to the dispatcher. How or if the Message Processing Subsystem modifies or utilizes the contents of the cache is message-processing-model-specific.

This may sound underspecified, but keep in mind that a message processing model might have access to all the information from the cache and from the message, and have no need to call a Security Model to do any processing; an application might choose a Security Model such as USM to authenticate and secure the SNMP message, but also utilize a secure transport such as that provided by the SSH Transport Model to send the message to its destination.

6.3. The `receiveMessage` ASI

If one does not exist, the Transport Model might create a cache referenced by `tmStateReference`. If present, this information might include `transportDomain`, `transportAddress`, `securityLevel`, and `securityName`, plus model or mechanism-specific details. How this information is determined is implementation and transport-model-specific.

This may sound underspecified, but keep in mind that a transport model might be something like SNMP over UDP over IPv6, where no security is provided, so it might have no mechanisms for determining a `securityName` and `securityLevel`.

The Transport Model does not know the `securityModel` for an incoming message; this will be determined by the Message Processing Model in a

[illegible]


```
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
```

The `tmStateReference` parameter of `prepareDataElements` is passed from the dispatcher to the Message Processing Subsystem. How or if the Message Processing Subsystem modifies or utilizes the contents of the cache is message-processing-model-specific.

The `processIncomingMessage` ASI passes `tmStateReference` from the Message Processing Subsystem to the Security Subsystem.

If `tmStateReference` is present and valid, an appropriate Security Model might utilize the information in the cache. How or if the Security Subsystem utilizes the information in the cache is security-model-specific.

This may sound underspecified, but keep in mind that a message processing model might have access to all the information from the cache and from the message, and have no need to call a Security Model to do any processing. The Message Processing Model might determine that the USM Security Model is specified in an SNMPv3 message header; the USM Security Model has no need of values in the `tmStateReference` cache to authenticate and secure the SNMP message, but an application might have chosen to use a secure transport such as that provided by the SSH Transport Model to send the message to its destination.

7. Security Considerations

This document defines an architectural approach that permits 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. Each proposed Transport Model should 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, implementers should store this information (in memory or in persistent storage) in a manner to protect it from unauthorized disclosure and/or modification.

Care must be taken to ensure that a SNMP engine is sending packets out over a transport using credentials that are legal for that engine to use on behalf of that user. Otherwise an engine that has multiple transports open might be "tricked" into sending a message through the wrong transport.

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, David Harrington, Keith McCloghrie, Kaushik Narayan, David 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 performed detailed reviews: Jeffrey Hutzelman, Bert Wijnen, Tom Petch.

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.
-----------	--

- [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.
- [RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer

- Security (TLS) Protocol Version 1.1", [RFC 4346](#), 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.
- [RFC4741] Enns, R., "NETCONF Configuration Protocol", [RFC 4741](#), December 2006.
- [I-D.ietf-isms-transport-security-model] Harrington, D., "Transport Security Model for SNMP", draft-ietf-isms-transport-security-model-03 (work in progress), February 2007.

[Appendix A](#). Parameter Table

Following is a Comma-separated-values (CSV) formatted matrix useful for tracking data flows into and out of the dispatcher, Transport, Message Processing, and Security Subsystems. This will be of most use to designers of models, to understand what information is available at which points in the processing, following the [RFC3411](#) architecture (and this subsystem). 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 line lengths.

[todo] this table needs to be updated.

[A.1](#). ParameterList.csv

```
,Dispatcher,,,Messaging,,Security,,Transport,
,sendPDU,returnResponse,processPDU,processResponse,
prepareOutgoingMessage,prepareResponseMessage,prepareDataElements,
generateRequest,processIncoming,generateResponse,
```


sendMessage, receiveMessage

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.

There are four approaches that could be used for passing information between the Transport Model and a 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) 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 Message Processing Model. The message header would be included in the wholeMessage ASI parameter, and would be removed by a corresponding Message Processing 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] defines 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 Message Processing 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 might 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

NOTE to RFC editor: If this section is empty, then please remove this open issues section before publishing this document as an RFC. (If it is not empty, please send it back to the editor to resolve.

- o MUST responses go back on the same transport session? How is this accomplished since the TM does not know `pduType`, and subsystems with knowledge of `pduType` do not know about transport sessions.

- o Do Informs work correctly?

[Appendix D](#). Change Log

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

Changes from -07- to -08-

- o Identified new parameters in ASIs.
- o Added discussion about well-known ports.

Changes from -06- to -07-

- o Removed discussion of double authentication
- o Removed all direct and indirect references to pduType by Transport Subsystem
- o Added warning regarding keeping sensitive security information available longer than needed.
- o Removed knowledge of securityStateReference from Transport Subsystem.
- o Changed transport session identifier to not include securityModel, since this is not known for incoming messages until the message processing model.

Changes from revision -05- to -06-

mostly editorial changes

removed some paragraphs considered unnecessary

added Updates to header

modified some text to get the security details right

modified text re: ASIs so they are not API-like

cleaned up some diagrams

cleaned up [RFC2119](#) language

added section numbers to citations to [RFC3411](#)

removed gun for political correctness

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 intro (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 receiveMessage ASI to use origin transport domain/address

updated Parameter CSV [appendix](#)

[C](#)hanges 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-

- o removed session table from MIB module
- o removed sessionID from ASIs
- o reorganized to put ASI discussions in EOP section, as was done in SSHSM
- o changed user auth to client auth
- o changed tmStateReference to tmSessionReference
- o 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-

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

Changes from revision -00-

- o changed SSH references from I-Ds to RFCs
- o removed parameters from tmSessionReference for DTLS that revealed lower layer info.

- o Added TMS-MIB module
- o Added Internet-Standard Management Framework boilerplate
- o Added Structure of the MIB Module
- o Added MIB security considerations boilerplate (to be completed)
- o Added IANA Considerations
- o Added ASI Parameter table
- o Added discussion of Sessions
- o Added Open issues and Change Log
- o 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

Copyright (C) The IETF Trust (2007).

This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgement

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

